조작체계

김책공업종합대학출판사
주체91

조작체계

김책공업종합대학출판사

차　례

제 2 편. 프로쎄스

제 3 장. 프로쎄스의 서술과 조종

제 4 장. 스레드, SMP 및 마이크로핵심부

제 5 장. 병행성 : 호상배제와 동기화

제 6 장. 병행성 : 교착과 고갈

제 3 편. 기억기

제 7 장. 기억기관리

제 8 장. 가상기억기

제 4 편. 일정작성

제 9 장. 단일처리기의 일정작성

제 10 장. 다중처리기와 실시간일정작성

제 5 편. 입출력 및 파일

제 11 장. 입출력관리와 디스크일정작성

제 12 장. 파일관리

제 6 편. 분산체계

제 13 장. 분산처리, 의뢰기/봉사기 및 클라스터

제 14 장. 분산형프로쎄스의 관리

제 7 편. 보안

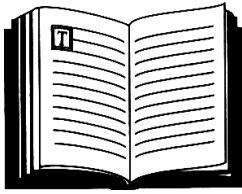
제 15 장. 보안

조작체계의 Web 싸이트; 해설 및 설계원리

William Stallings. com/OS4e.html에 있는 Web싸이트는 책을 사용하는 강사들과 대학생들에게 도움을 주기 위한것이다. 그것은 다음과 같은 내용들을 포괄한다.

교재류

교재류로서는 다음과 같은것들이 있다.

- 책의 그림들의 PDF복사물
- 책의 표들의 PDF복사물
- Java와 읽기 쉬운 Pascal과 같은 표준코드에 의한 모든 알고리듬의 PDF복사물
- 쉽게 참고하기 위하여 두개의 PDF서류로 만든 Windows와 UNIX의 모든 자료
- 강의의 보조수단으로 사용하기 위한 Power Point투영편묶음
- 쓸모 있는 학습방조를 줄수 있는 HTML로 된 강의록
- 콤퓨터과학부문 대학생 지원싸이트: 대학생들이 자기들이 받고 있는 콤퓨터과학교육에서 쓸모 있는것들을 찾을수 있는 많은 련결지령들과 서류들을 가지고 있다. 싸이트는 기초개괄, 련관된 수학;연구, 집필 및 련습문제풀이에 대한 조언;보고서보관 및 도서들과 같은 콤퓨터과학연구자원들에 대한 련결지령;그밖의 다른 유용한 련결지령들을 가지고 있다.
- 대체로 월마다 갱신되는 책의 고침표

OS 학과

OS4e Web싸이트는 이 책을 사용하여 가르치는 학과에서의 Web싸이트에 대한 련결지령을 가지고 있다. 이 싸이트는 일정작성 및 제목순서달기는 물론 많은 참고서와 다른 자료들에 대한 유익한 견해를 줄수 있다.

유용한 Web싸이트

OS4e Web싸이트는 련관된 Web싸이트들에 대한 련결지령을 가지고 있다. 련결지령은 많은 제목들을 포괄하고 있으며 대학생들이 더 심도 있게 시기적절한 문제들을 해명할수 있게 한다.

인터네트우편목록

인터네트의 우편목록은 이 책을 사용하는 강사들이 서로 또는 저자와 정보, 의견 및 질문을 교환할수 있게 한다. 예약정보는 책의 Web싸이트에 주어 져 있다.

조작체계의 도구

Web싸이트는 Nachos, OSP 및 BACI Web싸이트들에 대한 련결지령을 가지고 있다. 이것은 세가지 쏘프트웨어제품으로서 프로젝트실현에서 기본틀거리로 된다. 매개 싸이트는 내리적재할수 있는 쏘프트웨어 및 기초지식정보를 가지고 있다. 더 알고 싶은 문제가 있으면 부록 3을 보면 된다.

머 리 말

이 책의 목적

이 책은 조작체계의 개념, 구조 및 기구들을 서술한다. 이 책의 목적은 현 시대의 조작체계에 대한 본질과 특성지표를 될수록 명백하고 완전하게 보여 주자는데 있다.

이것은 몇가지 리유로 하여 복잡한 과제로 된다. 우선 조작체계의 설계범위가 상당히 넓고 콤퓨터체계가 다종다양한것이다. 이것은 단일사용자워크스테이션들과 개인용콤퓨터들, 중간규모의 공유체계들, 대형콤퓨터들과 초고속콤퓨터 및 실시간체계들과 같은 전용기계들을 포함하고 있다. 다양성은 기계의 용량이나 속도에서만이 아니라 응용프로그람들이나 체계지원요구사항들에서도 마찬가지이다. 다음으로 콤퓨터체계의 변화속도가 끊임없이 계속 높아 지고 있는것이다. 조작체계설계에서 몇가지 주요령역은 최근에 생겨 난 령역과 그것과 관련된 연구이며 다른 새로운 령역들이 계속 개척되고 있는것이다.

조작체계가 다양해 지고 변화가 있음에도 불구하고 일정한 기본적개념들은 전반에 걸쳐 확고하게 적용되고 있다. 확실한것은 이 개념들에 대한 응용이 현재의 기술상태와 특정한 응용요구사항들에 의존한다는것이다. 이 책의 취지는 조작체계설계의 기본에 대하여 전반적으로 설명하는것이며 이것을 이전의 설계문제들과 조작체계개발에서의 현재 방향들을 련관시키는데 있다.

실례체계

이 책은 독자들에게 조작체계의 설계원리와 실현문제를 습득시키는것을 목적으로 하고 있다. 따라서 순수 개념적이거나 리론적인 취급은 적합하지 않다고 본다. 개념을 설명하면서 해결해야 할 실제 세계의 설계문제들과 그것을 결부시키기 위해서 실행실례들로서 두개의 조작체계를 선택하였다.

- **Windows 2000**: 개인용콤퓨터, 워크스테이션들 및 봉사기들을 위한 다중과제조작체계이다. 새로운 조작체계로서 최근에 개발된 많은 수법들을 그대로 병합시킨것이다. 더우기 Windows 2000은 객체지향설계원리에 많이 의거하고 있는 실용화된 첫 조작체계들의 하나이다.
- **UNIX**: 원래 미니콤퓨터를 지향 했던 다중사용자조작체계이지만 성능 높은 극소형콤퓨터로부터 초고속콤퓨터에 이르기까지 넓은 령역에서 실현되였다. UNIX의 변종에는 세가지가 있다. UNIX SVR 4는 많은 기교특징들을 결합한 체계에 널리 쓰이고 있다. Solaris는 가장 널리 쓰이는 UNIX의 판본이다. Solaris는 다중스레드처리방식으로서 SVR 4와 대부분 다른 UNIX류형들에서 찾아 볼수 없는 기능들을 가지고 있다. 끝으로 Linux는 대중속에 급속히 보급되고 있는 UNIX의 원천공개판본이다.

그것들의 적합성과 표현성으로부터 이 체계를 선택하였다. 실례로 든 체계에 대한 설명은 한개의 장이나 부록과 같은 부속적으로가 아니라 본문전반에서 하고 있다. 그러므로 실례로 든 매개 체계의 병행성에 대하여 취급할 때 병행성수법들을 설명하며 개별적인 설계실례들을 선택하여 취급한다. 이러한 방법으로 주어 진 장들에서 설명한 설계개념들은 곧 실제 세계의 실례들로 다시 공고화시킨다.

독자대상

이 책은 대학생 및 전문가들을 독자대상으로 하고 있다. 교과서로서 콤퓨터과학, 콤퓨터공학 및 전기공학을 전공하는 대학생들을 위한 조작체계과목을 한 학기동안 취급한다. 책은 Computer Science and Engineering에서 IEEE Model Program의 5개 주제령역에 있는 제목들을 포괄하며 또한 Computer Science에서 Undergraduate Program의 ACM권고 CS 6 및 CS 10안에 있는 조작체계와 관련된 제목들을 포괄한다. 책은 또한 기초참고서로 될수 있으며 자체로 학습하기에 알맞는다.

책은 설명을 명백하게 하기 위하여 많은 그림과 표들을 주고 있으므로 명백히 교재적인 특징들을 가지고 있다. 더우기 책은 폭 넓은 용어해설, 흔히 쓰이는 략어 및 도서목록을 주고 있다. 매개 장에서는 련습문제들과 필요한 풀이방향들을 주고 있다. 매개 장에서는 또한 복습을 위해서 기본용어 및 여러개의 복습문제들을 제시하고 있다.

강사와 대학생들을 위한 인터네트봉사

대학생들과 강사들을 방조하는 이 책의 Web싸이트가 있다. 이 싸이트는 다른 해당한 싸이트들에 대한 련결지령들, PDF(Adobe Acrobat)형식으로 된 그림들과 표들 그리고 책의 인터네트우편목록을 위한 서명정보를 가지고 있다. Web페지는 William Stallings. Com/os4e. html인데 이 머리말의 앞에 있는《조작체계의 Web싸이트 해설 및 설계원리》부분을 보기 바란다. 인터네트우편목록은 이 책을 사용하는 강사들사이 또는 저자와 정보, 의견 및 질문을 교환할수 있도록 하기 위해 설정하였다. 인쇄오유와 다른 오유들을 발견하면 즉시에 William Stallings.Com에 있는 이 책의 고침표목록을 보면 된다.

조작체계학과의 실습과제

많은 강사들에게 있어서 조작체계학과의 중요한 구성요소는 실습과제와 실습과제의 모임인데 그것으로 대학생들이 참고할수 있는 지식을 얻고 본문의 개념들을 공고화한다. 이 책은 학과에서 실습과제부분을 포함하여 수많은 지원을 주고 있다. 실습과제실현에서 기본으로 되고 있는 세개의 쏘프트웨어제품 즉 조작체계의 구성요소를 개발하기 위한 OSP와 Nachos 그리고 병행성수법을 학습하기 위한 BACI의 지식을 주고 있다. 더우기 강사의 지도안에는 소규모의 프로그람작성실습과제들이 있는데 매개는 한주 또는 두주로 타산되여 있다. 그것은 넓은 문제범위를 포괄하고 있고 임의의 가동환경상에서 어떤 적합한 언어로 실현될수 있으며 또한 연구과제 및 문헌연구/보고서작성과제 등을 제시하고 있다. 구체적인것은 부록을 보기 바란다.

이 판에서 새로운 점

이 책의 전판(제3판)이 출판된후 3년안에 이 분야에서는 계속 발명과 개선이 이루어 졌다. 이 새로운 판에서는 총적인 분야를 포괄적으로 담으면서 이 변화를 반영하려고 하였다. 새 판을 준비하기 위하여 이 책의 전판은 이 과목을 강의하는 여러명의 교수들과 이 분야에서 사업하는 전문가들의 전면적인 후열을 받았다. 그 결과 많은 곳에서 해설이 명백해 졌고 확고해 졌으며 설명이 개선되였다. 또한 많은 새로운 분야의 문제들이 추가되였다.

교수자들과 사용자들에 대한 친절성을 도모하기 위한 노력을 거쳐 책의 기술적내용이 전반적으로 갱신되였으며 이 분야에서의 변화들을 반영하게 되였다.

제 0 장. 소개

이 책과 동반되는 Web싸이트는 많은 자료를 포괄하고 있다. 여기서는 독자들에게 개괄적으로 소개하려고 한다.

제 1 절. 책의 개요

이 책은 7개의 편으로 구성되여 있다.

제 1 편. 기초지식 : 콤퓨터구성방식과 조직에 대한 개요를 조작체계설계와 관련된 문제들에 중점을 두고 설명하며 책의 뒤부분에서 조작체계문제들에 대한 개요를 준다.

제 2 편. 프로쎄스 : 프로쎄스다중스레드처리, 대칭다중처리(SMP) 및 마이크로핵심부에 대한 구체적인 분석을 준다. 이 편에서는 또한 단일체계상에서 병행성에 대한 주요문제를 호상배제와 교착문제에 중점을 두고 론의한다.

제 3 편. 기억기 : 가상기억기를 포함한 기억기관리수법을 포괄적으로 고찰한다.

제 4 편. 일정작성 : 프로쎄스일정작성의 여러가지 방법들을 서로 비교하면서 설명한다. 스레드의 일정작성, SMP의 일정작성 및 실시간일정작성들에 대해서도 고찰한다.

제 5 편. 입출력과 파일 : 입출력기능을 조작체계의 조종밑에 실현하는데서 제기되는 문제점들을 론의한다. 체계성능측면에서 주요한 디스크입출력에 특별한 주의를 돌린다. 또한 파일관리에 대해서도 고찰한다.

제 6 편. 분산체계 : TCP/IP, 의뢰기/봉사기콤퓨터작업 및 클라스터를 포함하는 콤퓨터체계망화에 대한 기본추세를 론의한다. 또한 분산조작체계개발의 몇가지 주요설계령역을 서술한다.

제 7 편. 보안 : 콤퓨터 및 망보안을 실현하는데서 제기되는 위협과 수법들에 대하여 개괄적으로 고찰한다.

이 책의 취지는 해당 시기 조작체계들의 설계원리와 실현문제들에 정통하도록 하자는데 있다. 따라서 순수한 개념적 및 리론적 취급은 적합하지 않다. 개념들을 설명하고 그것을 실제적인 설계물들과 결부시키기 위하여 두개의 조작체계를 실행실례로 선정하였다.

- Windows 2000 : 여러가지 PC, 워크스테이션 및 봉사기상에서 실행하도록 설계된 다중과제처리조작체계이다. 이것은 최근에 상품화된 몇가지 조작체계들중의 하나로서 본질상 특수하게 설계된것이다. 그러므로 조작체계수법에서 가장 최근에 개발된 새형의 조작체계의 지위를 차지하고 있다.
- UNIX : 초기에 소형콤퓨터에서 시도했으나 성능이 높은 극소형콤퓨터로부터 초고속콤퓨터에 이르기까지의 넓은 기계령역에서 실현시킨 다중과제처리조작체계이다.

이 실례체계들에 대하여서는 한개의 장이나 부록으로가 아니라 책전반에 걸쳐 설명하고 있다. 그러므로 병행성에 대한 설명을 하면서 매 실례체계들의 병행성기구를 서술

하며 개별적인 설계선택동기를 설명하게 된다. 이러한 방법으로 주어 진 장들에서 론의된 설계개념들은 곧 현실실례들과 결부된다.

제 2 절. 제목순서달기

독자들이 이 책에서 보여 준 문제들에 특정한 순서달기를 론하는것은 자연스럽다. 실례로 일정작성문제(제9장과 제10장)는 병행성(제5장과 제6장)의 일정작성문제와 밀접히 련관되여 있고 프로쎄스의 일반문제들(제3장)은 그밖의 문제들을 합리적으로 포괄하고 있다.

난점은 여러가지 문제들이 호상 깊이 련관되여 있다는것이다. 실례로 가상기억기를 설명하는데서 페지부재와 관련되는 일정작성문제를 같이 언급하는것이 좋다. 물론 일정작성결정을 설명할 때 일정한 기억기관리 즉 일정작성문제를 입출력관리와 장치와 함께 리해하거나 또는 그 반대의 경우에도 함께 리해할것을 요구한다.

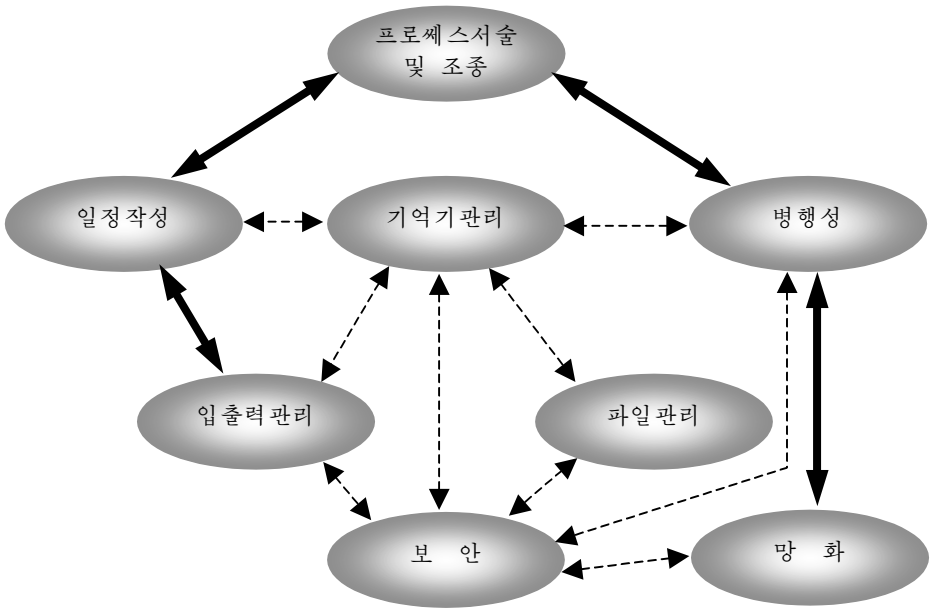



그림 0-1. 조작체계들에서 제목들사이의 관계

그림 0-1은 제목들사이에서 중요 호상관계를 보여 준다. 굵은 선들은 설계와 실현결정의 견지에서 매우 밀접한 관계를 보여 준다. 이 그림을 보면 실제로 제3장에서 할 프로쎄스에 대한 기초적인 설명을 시작하는것 같다. 그다음의 순서는 어느 정도 독단적이다. 많은 조작체계취급에서도 먼저 프로쎄스들의 모든 요소를 갈라 놓은 다음 다른 문제를 론한다. 이것은 아주 효과적이다. 그러나 프로쎄스관리와 꼭 같은 중요성을 가진다고 보는 기억기관리의 주요한 의의는 일정작성을 심도있게 보기에 앞서 이 요소들을 보여줄것을 결심하게 하였다.

리상적인 해결방도는 학구적인 사람이 제1장부터 제3장을 련이어 읽은 다음에 다음장들을 병렬로 읽고 소화하는것이다. 즉 제5장(선택적으로)을 읽고 제4장, 제7장을 읽고 제6장, 제9장(선택적으로)을 읽고 제8장, 제10장을 읽는 식으로 읽어야 소화할수 있다. 끝

으로 다음의 장들은 임의의 순서로 읽을수 있다. 즉 제11장, 제13장을 읽고 제12장, 제14장, 제15장을 읽을수 있다. 그러나 사람의 뇌수가 병렬처리를 할수 있다고 해도 사람의 학구력은 같은 책에서 4개의 서로 다른 장들을 동시에 펼치고 그 내용을 동시에 완전히 리해한다는것은 불가능하다는것을 보여 주고 있다. 따라서 순서달기의 필요성이 제기된 조건에서 이 책에서의 순서달기가 가장 효과적이라고 본다.

제 3 절. 인터네트 및 Web자원

인터네트와 Web상에는 받아 들일수 있는 많은 자원들이 있어 이 책을 지원하고 있고 이 분야에서의 개발을 계속할수 있게 도와 주고 있다.

이 책의 Web싸이트

William Stallings.com/OS4e.html로서 이 책을 위한 특별한 Web싸이트를 설정해 놓았다. 학생들을 위한 Web싸이트에서 받아 들일수 있는 많은 서류들중에는 특정한 표기들이 있다.

- **Java코드 :** 책전반에 걸쳐 많은 코드들이 있는데 그것은 C에서 여러가지 OS와 관련된 알고리듬을 정의한다. 이 모든 코드들은 Web싸이트에 있는 Java에서 재생할수 있다.
- **Java프라이머 :** 보다 흥미를 가지는 학생들을 위해 간단한 Java프라이머를 준다.
- **Pseudocode :** C나 Java에 익숙되지 못한 독자들을 위해 모든 알고리듬을 Pascal과 같은 류사코드로 재생시킨다. 이 류사코드언어는 직관적이며 특히 리해하기 쉽다.
- **Windows 2000과 UNIX서술 :** 이미 언급한바와 같이 Windows 2000과 여러가지 UNIX판본들이 실행중인 경우의 연구에 사용되고 있는데 그 설명은 한개 장이나 부록으로서가 아니라 본문전반에 걸쳐 진행되고 있다. 일부 독자들은 이 모든 자료를 한 곳에서 참조하려고 할수도 있다. 그러므로 이 책에서 보게 되는 Windows와 UNIX의 모든 자료가 Web싸이트에 2개의 서류로 재생되여 있다.

임의의 인쇄오유나 다른 오유를 발견하면 이 책의 오유목록을 Web싸이트에서 찾아 볼수 있다. 발견한 오유들을 보고해 주기 바란다. 이 책에서의 순서달기정보는 물론 고려하지 않고 같은 필자가 쓴 다른 책들에서의 틀린 글자판들도 William Stallings.com에 있다.

필자는 또한 William Stallings.com/StudentSupport.html에 Computer Science Student Support Site를 가지고 있다. 이 싸이트의 목적은 콤퓨터과학부문학생들을 위한 서류, 정보 및 련결지령을 주는데 있다. 련결지령은 4개의 범주로 구성되여 있다.

- **Math :** 기초수학상기프로그람, 대기처리분석프라이머, 수체계프라이머 및 많은 수학싸이트들에 대한 련결지령
- **How-to :** 숙제문제들을 풀며 기술보고서 작성 및 기술공개를 준비하는데서의 방법적조언과 안내
- **Research resources :** 중요한 론문, 기술보고서, 도서목록묶음에 대한 련결지령
- **Miscellaneous :** 여러가지 유용한 서류와 련결지령

다른 Web싸이트

이 책과 관련된 일정한 종류의 정보를 주는 많은 Web싸이트들이 있다. 다음과 같은 것들을 들수 있다.

- UNIX Reference Desk : UNIX에 대한 정보, 그밖의 제품, 사용자대면부 및 기술정보에 대한 지침
- UNIX Guru Universe : UNIX정보의 다른 좋은 원천
- Linux Documentation Project : 싸이트를 서술하는 이름
- OS Web : Multics와 같은 고전적인것을 포함하는 많은 조작체계들에 대한 정보
- ACM Special Interest Group on Operating Systems : SIGOPS출판 및 토론회에 대한 정보
- IEEE Technical Committee on Operating Systems and Application Environments : 직결시사해설과 다른 싸이트들에 대한 련결지령을 가지고 있다.
- IEEE Computer Society Task Force on Cluster Computing : 클라스터계산연구 및 교육을 촉진시키기 위한 국제토론회
- The comp.os.research FAQ : 조작체계설계문제를 포괄하는 구체적이며 가치있는 FAQ
- The Memory Management Reference : 기억기관리의 모든 문제들에 대한 훌륭한 서류와 련결지령의 좋은 원천
- The Operating System Resource Center : 넓은 범위의 OS문제들에 대한 서류와 론문들의 쓸모 있는 묶음

Web싸이트의 주소가 자주 변화되는 경향이 있으므로 여기에 포함시키지 않았으며 적당한 련결지령은 이 책의 Web싸이트에서 찾아 볼수 있다.

USENET 새 그룹

많은 USENET 새 그룹들이 조작체계의 일부 문제 또는 특별한 조작체계에 기여하고 있다. 사실상 모든 USENET그룹들을 놓고 보면 높은 신호대잡음비가 있지만 그것이 사람들의 요구를 만족시키겠는가를 실험해 보아야 한다. 다음과 같은 두개가 가장 적절하다고 본다.

- Comp.os.research : 따라 가는데 가장 좋은 집단. 이것은 연구과제를 취급하는 완만한 새 그룹이다.
- Com.os.misc : OS문제들에 대한 일반적인 설명

제 1 편. 기초지식

제 1 편의 중심

이 편에서는 이 책에서 보게 되는 조작체계에 대한 기초지식과 전후관계를 서술한다. 또한 콤퓨터구성과 조작체계의 기본개념들을 보여 준다.

제 1 편의 안내

제1장. 콤퓨터체계의 개괄

조작체계는 응용프로그람들과 편의프로그람들, 사용자들을 한편으로 하고 콤퓨터체계장치를 다른 한편으로 하여 그사이에서 가동한다. 조작체계의 기능과 설계문제들을 인식하기 위해서는 콤퓨터구성과 방식에 대한 일정한 지식을 가져야 한다. 제1장에서는 콤퓨터체계의 처리기, 기억기와 입출력요소들에 대한 주요개념들을 설명한다.

제2장. 조작체계의 개괄

조작체계설계는 많은 문제를 포괄하고 있으며 세부적인 측면들을 놓치거나 특정한 문제들에 대한 론의에서 전후관계를 헛갈리기가 아주 쉽다. 제2장은 독자들이 전후관계를 임의의 곳에서 알아 볼수 있게 하는 내용을 개괄하고 있다. 먼저 조작체계의 객체들과 기능들에 대한 설명을 준다. 다음 력사적으로 중요한 조작체계들과 그 기능들을 설명한다. 이러한 설명은 단순한 환경에서 몇가지 기본 OS설계원리들을 보여 줌으로써 여러가지 OS기능들사이의 관계를 명백하게 해준다. 이 장에서는 또한 현대적인 조작체계들의 중요한 특징들을 집중적으로 고찰한다. 이 책의 전반적인 부분에서는 여러가지 문제들을 설명하는데 기본적이면서 잘 확립된 원리들은 물론이고 OS설계에서 보다 최근에 발명된 내용들도 모두 설명한다. 독자들은 이미 확립되여 있는 설계수법들과 여기서 설명하는 최근 설계수법들을 혼탁시키지 말아야 한다. 끝으로 Windows 2000과 UNIX에 대하여 개괄하는데 이것은 이 체계들의 일반적구조를 확고하게 해준다.

제 1 장. 콤퓨터체계의 개괄

조작체계는 하나이상의 처리기들로 구성되여 있는 장치자원들을 개발하여 체계사용자들에게 어떤 봉사를 준다. 조작체계는 또한 사용자들에게 편리하게 2차기억기와 입출력장치들을 관리한다. 그러므로 조작체계에 대한 연구를 시작하는 견지에서 체계장치들에 대한 일정한 리해를 가져야 한다.

이 장에서는 콤퓨터체계장치를 개괄적으로 고찰한다. 독자들이 이 학문분야에 관련이 있다고 보면서 대부분 중점적으로 고찰하였다. 그러나 일부 부분들에서는 후에 보게되는 문제들의 중요성으로부터 좀 더 상세히 고찰하였다.

제 1 절. 기본구성요소

웃준위에서 볼 때 콤퓨터는 처리기, 기억기와 입출력구성요소들로 이루어 져 있으며 매개 요소들은 하나이상의 모듈들을 가지고 있다. 이 구성요소들은 일정한 방식으로 호상 련결되여 콤퓨터의 주요기능을 구성하며 이것이 프로그람들을 집행하게 된다. 이로부터 4개의 기본구조적요소들이 있다.

- **처리기 :** 콤퓨터의 동작을 조종하며 자료처리기능을 수행한다. 한개의 처리기만을 사용할 때 그것을 흔히 중앙처리장치라고 한다.
- **주기억기 :** 자료와 프로그람들을 기억한다. 이 기억기는 대체로 휘발성이며 실기억기 또는 1차기억기라고도 부른다.
- **입출력모듈 :** 콤퓨터와 외부환경사이에서 자료를 이동시킨다. 외부환경은 각이한 외부장치들로 구성되는데 2차기억장치, 통신설비와 말단들을 포함한다.
- **체계모선 :** 처리기들과 주기억기 그리고 입출력모듈들사이에서 통신을 보장하여 주는 일정한 구조와 기구들이다.

그림 1-1은 웃준위구성요소들을 보여 주고 있다. 처리기의 한가지 기능은 기억기와 자료교환을 하는것이다. 이로부터 대체로 2개의 내부(처리기에 대하여)등록기를 사용한다. 하나는 기억주소등록기(MAR)로서 다음번에 읽거나 써야 할 기억기의 주소를 지정한다. 다른 하나는 기억완충등록기(MBR)로서 기억기에 써넣어야 할 자료 또는 기억기로부터 읽어 낸 자료를 포함한다. 이와 류사하게 입출력주소등록기(I/OAR)는 특정한 입출력장치를 지정한다. 입출력완충등록기(I/OBR)는 입출력모듈과 처리기사이에서 자료교환용으로 쓰인다.

기억기모듈은 련속적으로 수자화된 주소로 지적되는 하나의 장치로 되여 있다. 매개 주소는 2진수로 되여 있는데 그것은 명령이나 자료중 어느 하나를 지적할수 있다. 입출력모듈은 외부장치들로부터 처리기와 기억기에로 또는 그 반대로 자료를 이송한다. 입출력모듈은 자료들을 이송할 때까지 림시적으로 그것을 보관하기 위한 내부완충기들을 가지고 있다.

제 2 절. 처리기의 등록기

처리기는 한조의 등록기들을 포함하고 있는데 이 등록기들은 주기억기보다 속도가 빠르면서 크기가 작은 어떤 준위의 기억기이다. 처리기내의 등록기들은 두가지 기능을 보장한다.

- **사용자변경등록기** : 기계어나 아쎔블리언어프로그람작성자들이 등록기사용을 최적화함으로써 주기억기참조를 최소화할수 있게 한다. 고급언어에서 번역기를 최적화하자면 등록기들에 할당하는 변수들과 주기억기주소들도 할당하는 변수들을 지능적으로 선택할수 있게 해야 할것이다. C와 같은 일부 고급언어들은 프로그람작성자들이 등록기들에 유지하려고 하는 변수들을 번역기에 제시하도록 하고 있다.
- **조종 및 상태등록기** : 이 등록기들을 사용하여 처리기는 처리기의 동작을 조종하며 프로그람들의 집행을 조종한다.

이 범주에 들어 가는 등록기들에 대한 명백한 구분은 없다. 실례로 일부 기계들에서 프로그람계수기는 사용자가 변경시킬수 있지만 많은 경우에 그렇지 못하다. 그러나 다음 단계의 설명을 위해서 이 범주들을 사용하면 편리하다.

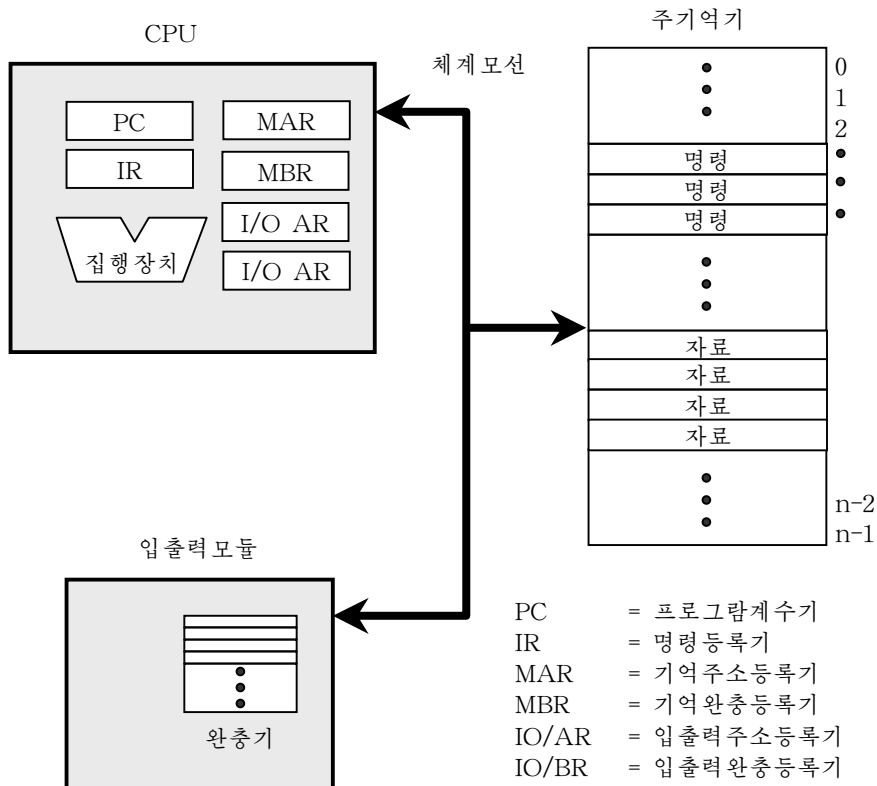



그림 1-1. 웃준위에서 본 콤퓨터구성요소들

사용자변경등록기

사용자변경등록기는 처리기가 집행하는 기계어에서 참조할수 있으며 일반적으로 응용프로그람들은 물론 체계프로그람들을 포함하는 모든 프로그람들에서 참조할수 있다. 대체로 사용할수 있는 등록기형들은 자료, 주소 및 조건부호등록기들이다.

자료등록기들은 프로그람작성자들이 함수에 대한 어떤 변수로 설정할수 있다. 일부 경우에 그것들은 본질상 일반목적이며 자료에 대한 조작을 수행하는 임의의 기계명령에

대해 쓸수 있다. 그러나 흔히 제한들이 있다. 실례로 류동소수점연산에는 전용등록기들을 쓸수 있고 옹근수연산에는 다른것들을 쓸수 있다.

주소등록기들은 자료와 명령들에 대한 주기억기주소를 담고 있거나 완전한 즉 실제적인 주소계산에 사용하는 일부 주소를 담고 있다. 이 등록기들은 그들자체가 일반목적용일수도 있고 또는 기억기를 주소화하는 특정한 방안이나 방식에서 사용할수도 있다. 다음과 같은 실례들을 들수 있다.

- **첨수등록기 :** 첨수주소화는 주소화의 일반적인 방법으로서 여기서는 하나의 기준값에 어떤 첨수를 더하여 실제적인 주소를 얻는다.
- **토막지시기 :** 토막주소화를 보면 기억기는 토막들로 분할되며 그 토막들은 가변길이를 가진 단어블로크들이다. 기억기참조는 특정한 토막과 그 토막내에 있는 편위에 대한 참조로 이루어 진다. 이 주소화방식은 제7장에서 설명하는 기억기관리에서 중요한 방식이다. 주소화방식에서 등록기는 토막의 기준주소(출발위치)를 유지하는데 쓰인다. 다중등록기들이 있을수 있는데 실례를 들면 조작체계용으로 한개(즉 조작체계코드가 처리기상에서 실행되고 있을 때)와 현재 응용프로그람 집행용으로 한개가 있을수 있다.
- **탄창지시기 :** 사용자변경탄창[1]주소화가 있다면 그 탄창의 꼭대기를 가리키는 전용등록기가 있다. 이것은 밀어넣기와 밀어내기와 같이 아무런 주소마당도 포함하지 않는 명령을 사용하게 한다.

일부 기계들에서 어떤 수속 즉 보조루틴호출은 있는 등록기들을 모두 자동적으로 보관하였다가 복귀할 때 회복될것이다. 보관과 회복은 호출과 복귀의 명령실행부분으로서 처리기가 수행한다. 이것은 매개 수속이 이 등록기들을 독립적으로 쓸수 있게 한다. 다른 기계들에서는 수속호출에 앞서 관련이 있는 사용자등록기들의 내용보관을 프로그람작성자가 하며 프로그람에 이 목적을 위한 명령을 포함하고 있다. 이와 같이 보관과 회복기능은 기계에 따라서 장치적으로 또는 프로그람적으로 수행될수 있다.

조종 및 상태등록기

처리기의 동작을 조종하기 위하여 여러가지 처리기의 등록기들을 사용한다. 많은 기계들에서 이 등록기들은 대부분 사용자가 변경시킬수 없다. 그들중 일부는 조종 또는 조작체계방식으로 귀착되는 기계명령을 집행하여 호출할수도 있다.

물론 각이한 기계들은 서로 다른 등록기방식들을 가지며 각이한 전문용어들을 쓴다. 우리는 여기서 합리적이며 완전한 등록기형들에 대한 목록을 기본적인 설명을 붙여 정리한다. 이미 언급된 MAR, MBR, I/OAR, I/OBR등록기들외에 명령집행에서 본질적인것들을 아래에 설명한다.

- **프로그람계수기(PC) :** 불러 내야 할 명령주소가 있다.
- **명령등록기(IR) :** 가장 최근에 불러 낸 명령이 있다.

모든 처리기설계는 프로그람상태단어(PSW)로 알려 져 있는 어떤 등록기나 등록기묶음을 포함하게 되는데 PSW는 상태정보이다. PSW는 대체로 조건코드들과 새치기가능/불가능비트와 관리기/사용자방식비트와 같은 다른 상태정보를 포함하고 있다.

조건부호들(기발이라고도 한다.)은 처리기장치의 조작결과에 얻어 진 비트모임이다.

---

[1]. 탄창은 주기억기에 있으며 련속적인 주소의 모임으로서 우에서부터 넣고 불러 냄으로써 물리적탄창과 류사하게 볼수 있다. 탄창처리에 대한 설명은 부록 1-ㄴ를 보시오.

실례로 산수연산은 정,부, 령이나 자리넘침결과를 산생시킬수 있다. 결과 그자체는 등록기나 기억기에 기억되는외에 조건부호는 그에 따르는 산수연산명령집행에 또 영향을 주게 된다. 부호는 조건분기조작의 일부로서 뒤따라 검사될수 있다. 조건부호비트들은 하나이상의 등록기들에 집중된다. 보통 그것들은 조종등록기부분을 형성한다. 일반적으로 기계명령들은 이 비트들을 암시적참조형식으로 읽어 내지만 명령집행결과를 보면 그 비트들은 귀환을 목적으로 하고 있기때문에 공개적참조형식으로 바꿀수 없다.

다중새치기형을 사용하는 기계들에서는 매개 새치기조종루틴에 하나의 지시기를 가지는 한조의 새치기등록기를 가질수 있다. 일정한 기능(실례로 수속호출)을 실현하기 위하여 탄창을 사용한다면 그때 체계탄창지시기를 써야 한다(부록 1-ㄴ를 보시오.). 제7장에서 설명하는 기억기관리장치는 전용등록기들을 요구한다. 끝으로 입출력조작들을 조종하는데 등록기들을 사용할수 있다.

조종 및 상태등록기방식설계에는 몇가지 인자들이 들어 간다. 한가지 주요문제점은 조작체계를 지원하는것이다. 조종정보의 정확한 형태는 조작체계에서 볼 때 전문프로그람인것이다. 처리기설계자가 사용되는 조작체계에 대한 기능적인 리해를 하고 있다면 등록기방식기억기보호와 사용자프로그람들사이의 절환과 같은 특정한 기능들을 장치적으로 지원할수 있도록 설계할수 있다.

다른 하나의 주요설계문제는 등록기들과 기억기사이에서 조종정보의 할당이다. 기억기단어의 수백 또는 수천분의 1은 조종목적에 사용된다. 설계자는 값이 비싸지만 속도가 빠른 등록기들에 얼마만한 조종정보를 할당하며 값이 눅지만 속도가 느린 주기억기에 어느만큼을 할당할것인가를 결심해야 한다.

제 3 절. 명령집행

처리기가 집행하여야 할 프로그람은 기억기에 기억된 명령의 모임으로 되여 있다. 가장 단순한 형태로서 명령처리는 두 단계로 즉 첫 단계에서 처리기는 어떤 시각에 기억기로부터 명령을 읽어 내며(불러내기) 둘째 단계에서 매개 명령을 집행한다. 프로그람집행은 명령불러내기와 명령집행의 반복과정이다. 명령집행은 몇개의 조작을 포함할수 있는데 그것은 그 명령의 특성에 관계된다.

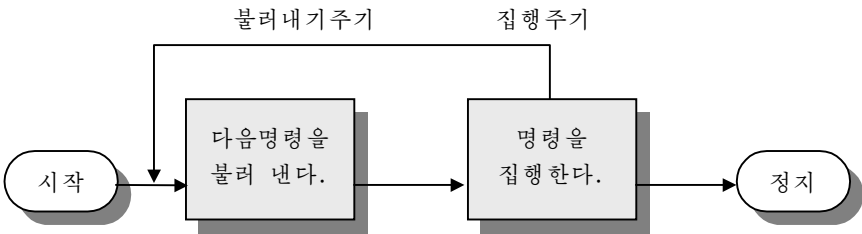



그림 1-2. 기본명령주기

하나의 단순한 명령에 요구되는 처리를 명령주기라고 한다. 간단한 두 단계설명을 사용하여 명령주기를 그림 1-2에 보여 주었다. 두 단계는 불러내기주기와 집행주기로 표시하였다. 프로그람집행은 기계에 전원이 차단되였거나 일부 회복할수 없는 오유가 발생하였을 때 또는 콤퓨터를 정지시키는 프로그람명령과 부닥쳤을 때에만 정지한다.

명령불러내기와 집행

매개 명령주기의 초기에 처리기는 기억기로부터 명령을 불러 낸다. 대표적인 처리기들에서 프로그람계수기(PC)는 다음번에 불러 내야 할 명령주소를 보관하고 있다. 다른

한편 다른 명령을 받지 않는 한 처리기는 매개 명령을 불러 낸후에 항상 PC를 증가시켜 다음명령(즉 다음의 높은 기억주소에 있는 명령)을 순서대로 불러 낼수 있다. 실례로 매개 명령이 하나의 16bit단어기억기를 차지하는 간단한 콤퓨터를 고찰하여 보자. 프로그람계수기는 위치 300에 설정되여 있다고 가정한다. 처리기는 위치 300에 있는 다음명령을 불러 낸다. 명령주기를 이어가면서 처리기는 위치 301,302,303,…으로부터 명령을 불러 낸다. 후에 설명하게 되겠지만 이 순서는 변화될수 있다.

불러 낸 명령을 명령등록기(IR)라고 하는 처리기안에 있는 등록기에 적재한다. 명령은 처리기가 취하게 될 동작을 규정하는 비트들을 포함한다. 처리기는 명령을 해석하고 요구되는 동작을 수행한다. 일반적으로 이 동작들은 4개의 범주에 귀착된다.

- **처리기-기억기 :** 처리기에서 기억기에로 또는 기억기에서 처리기에로 자료를 이송할수 있다.
- **처리기-입출력 :** 처리기와 입출력모듈사이의 이송방식으로 주변장치로부터 또는 주변장치에로 자료를 이송할수 있다.
- **자료처리 :** 처리기는 자료에 대한 일정한 산수적 또는 론리적연산을 수행할수 있다.
- **조종 :** 명령은 집행순서가 달라 진다는것을 지정할수 있다. 실례로 처리기는 주소 149에서 명령을 불러 낼수 있는데 그 명령은 다음명령이 주소 182에 있다는것을 지정할수 있다. 처리기는 PC를 182로 설정한다. 그러므로 다음 불러내기주기에서 150이 아니라 주소 182부터 명령을 불러 낸다.

그림 1-3. 가상기계의 특성: ㄱ-명령형식, ㄴ-옹근수형식,
ㄷ-내부 CPU등록기, ㄹ-연산코드의 부분목록(2진형식)

명령집행은 이 동작들의 조합으로 이루어 진다.

그림 1-3에 보여 준 특성을 가지는 가상기계를 사용하여 간단한 실례를 들어 보자. 처리기는 축적기(AC)가 호출한 하나의 자료등록기를 가지고 있다. 명령과 자료는 둘다 16bit길이를 가진다. 이렇게 되면 16bit 즉 단어를 사용하여 기억기를 조직하기 편리하다. 명령형식은 연산코드에 4bit를 할당하므로 2^4 =16개의 각이한 연산코드들이 있을수 있으며(한자리의 16진수[2]로 표시된다.) 단어기억기를 직접주소지정할수 있다(3자리의 16진수

2. 수체계(10진, 2진, 16진)는 the Computer Science Student Support Site at William Stallings.com/Student Support. html.에서 참조할수 있다.

로 표기된다.).

그림 1-4는 부분프로그람집행실례로서 기억기와 처리기의 등록기들을 보여 준다. 프로그람소편은 주소 940에 있는 기억기단어의 내용을 주소 941에 있는 기억기단어의 내용에 더하고 그 결과를 다음위치(주소 941)에 기억한다는것을 보여 주고 있다. 세개의 불러내기주기와 세개의 집행주기로 설명할수 있는 세개의 명령이 필요하다.

1. PC는 300을 가지는데 첫 명령의 주소이다. 이 명령(16진수로 1940)은 명령등록기 IR에 적재되고 PC는 증가한다. 이 처리에서 기억주소등록기(MAR)와 기억완충등록기(MBR)를 쓰는데 설명을 간단히 하기 위하여 표시하지 않았다.

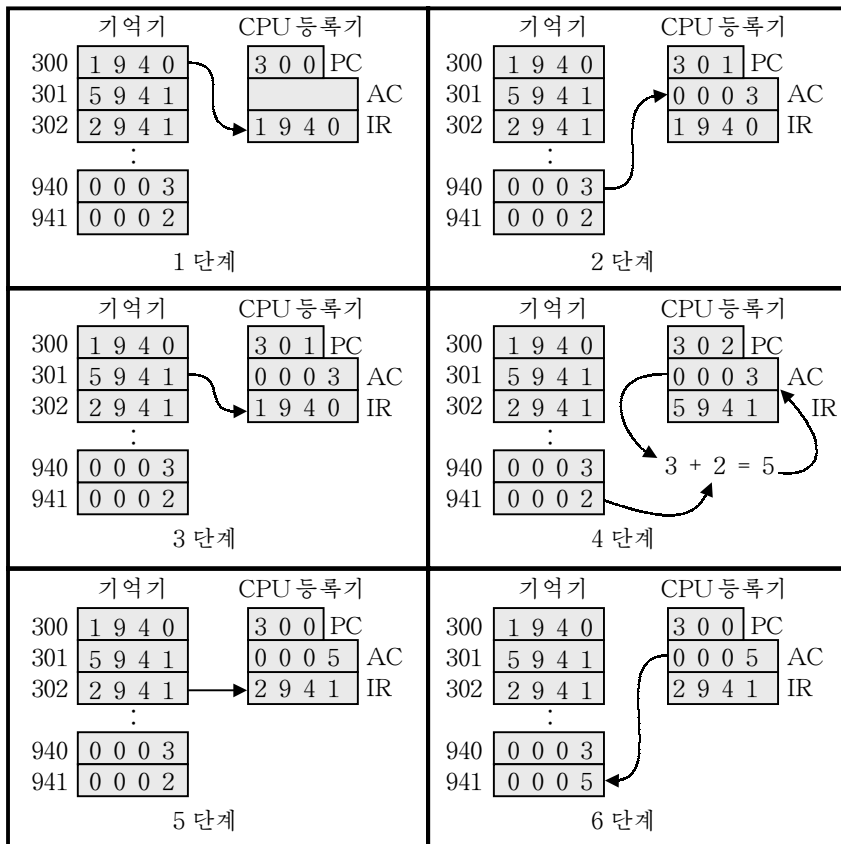



그림 1-4. 프로그람집행실례(16진수에서 기억기와 등록기들의 내용)

2. IR에서 첫 4bit(첫 16진수)는 AC가 적재된다는것을 가리킨다. 나머지 12bit(세개의 16진수)는 주소가 940이라는것을 지적한다.
3. 다음명령(5941)은 주소 301에서 불러 내며 PC는 증가한다.
4. AC에 이미 있던 내용과 주소 941의 내용이 더해 지고 결과가 AC에 기억된다.
5. 다음명령(2941)은 주소 302에서 불러 내며 PC는 증가한다.
6. AC의 내용이 주소 941에 기억된다.

이 실레에는 세개의 명령주기가 있는데 개개는 불러내기주기와 집행주기로 되여 있으며 주소 940의 내용을 841의 내용에 더할것을 요구한다. 더 복잡한 명령들에서는 주기가 좀 더 요구된다. 현대처리기들은 대부분 하나이상의 주소를 포함하는 명령들을 가지고 있다. 때문에 특정한 명령의 집행주기에는 한번이상의 기억기참조가 있게 된다. 또한 기억기참조대신에 명령은 입출력조작을 지정할수도 있다.

입출력기능

여기까지 우리는 처리기가 조종하는 콤퓨터의 동작을 설명하였고 처리기와 기억기에 대하여 초보적으로 보았다. 설명은 입출력구성요소의 역할에 대하여 스쳤을뿐이다.

입출력모듈(실례로 디스크조종기)은 처리기와 직접 자료를 교환할수 있다. 처리기가 기억기에 대한 읽기나 쓰기, 특수한 위치에 대한 주소지정을 할수 있는것과 꼭같이 입출력모듈로부터 자료를 읽거나 써넣을수 있다. 이 경우에 처리기는 특정한 입출력모듈이 조종하는 지정된 장치를 식별한다.

일부 경우에 입출력과제에 대한 처리기의 부담을 덜어 주기 위하여 입출력장치가 직접 기억기를 관리하게 하려는 요구가 있을수 있다. 그러한 경우에 처리기는 처리기에 매이지 않고 입출력-기억기사이에서 자료를 받을수 있다. 자료의 주고받기기간에 입출력모듈은 기억기에 읽기나 쓰기지령을 내보내며 자료교환에서 처리기의 부담을 덜어 준다. 이 동작을 직접기억기접근(DMA)이라고 하는데 후에 다시 고찰한다.

제 4 절. 새치기

실제상 모든 콤퓨터들은 처리기가 일반처리를 하는 경우 다른 모듈들(입출력, 기억기)이 새치기할수 있는 기구를 갖추고 있다. 표 1-1은 가장 일반적인 새치기들을 분류하여 놓은것이다.

표 1-1. 새치기의 분류

프로그람	산수적 자리초과, 0에 의한 나누기, 허용할수 없는 기계명령을 집행하려는 시도, 사용자에게 허용된 기억구역밖을 참조하려는것과 같은 명령집행결과로 하여 생기는 일부 조건들에 의하여 발생한다.
시계	처리기안에 있는 시계에 의하여 발생한다. 이것은 조작체계가 정규적인 기준우에서 정해 진 기능을 수행하도록 한다.
입출력	입출력조종기에 의하여 발생하는데 어떤 조작을 정상완료하였다는 신호로 되거나 각이한 오유조건에 대한 신호로 된다.
장치고장	장치고장이나 기억기부분오유와 같은 고장에 의해 발생한다.

새치기들은 일차적으로 처리효률을 개선하기 위한 한가지 방도로 된다. 실례로 대부분 입출력장치들은 처리기보다 속도가 훨씬 느리다. 이제 처리기가 그림 1-2의 명령주기도식을 사용하여 인쇄기에 자료를 전달한다고 가정해 보자. 매개 쓰기조작을 한후 처리기는 인쇄기가 붙잡을 때까지 정지해야 하며 휴식상태에 있어야 한다. 그 정지길이는 수백내지 수천개의 명령주기(기억기는 다치지 않는다.)에 이를수 있다. 이것은 처리기에 대해서 명백하게 랑비로 된다.

그림 1-5 ㄱ는 이와 같은 상태의 사건들을 보여 주고 있다. 사용자프로그람은 처리와 교차되여 련속적인 WRITE호출을 수행한다. 코드토막 1, 2와 3은 입출력을 포함하지 않

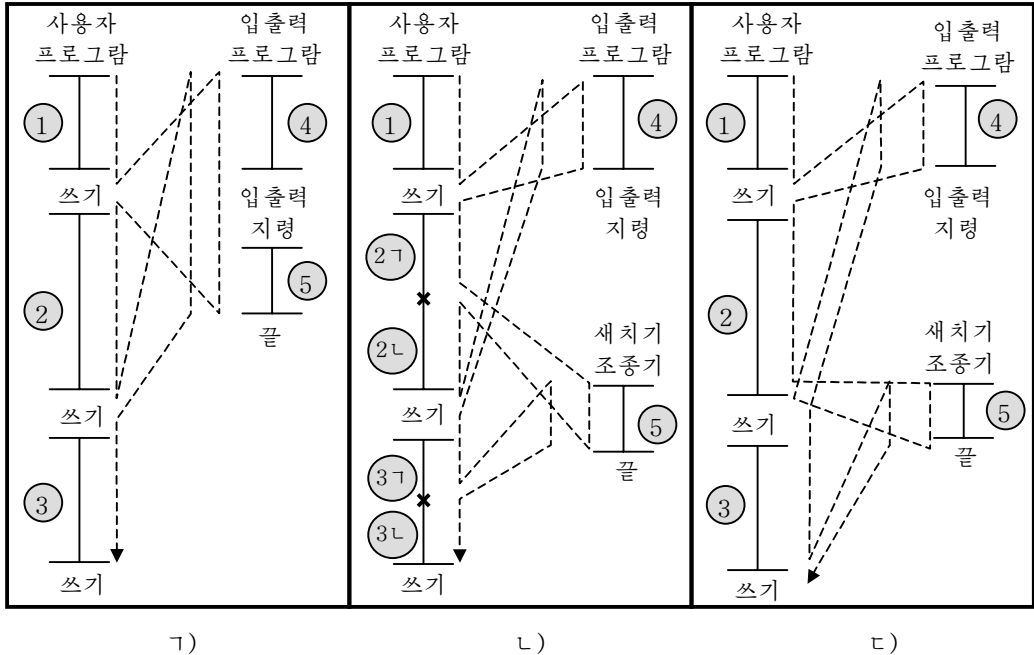



그림 1-5. 새치기여부에 관계 없는 프로그람조종흐름: ㄱ-새치기가 없다, ㄴ-새치기가 있고 입출력기다림이 짧다, ㄷ-새치기가 있고 입출력기다림이 길다

는 명령렬을 가리킨다. WRITE호출들은 체계편의응용프로그람과 실제적인 입출력동작을 수행할 입출력프로그람이다. 입출력프로그람은 세개의 부분으로 구성되여 있다.

- 그림에서 4로 표기하고 있는 명령렬로서 실제적인 입출력동작을 준비하기 위한것이다. 이것은 특정한 완충기에 출력해야 할 자료복사와 장치지령을 위한 파라메터준비를 포함할수 있다.
- 실제적인 입출력지령새치기를 사용하지 않고 일단 이 지령이 나오면 프로그람은 입출력장치가 필요한 기능을 수행하기를 기다려야 한다(또는 입출력장치에 대해 주기적인 상태검사, 문의조사를 해야 한다.). 프로그람은 입출력조작이 수행되였는가를 확인하기 위한 검사조작을 단순히 반복수행하면서 기다릴수 있다.
- 그림에서 5로 표기하고 있는 명령렬로서 조작을 완성하기 위한것이다. 이것은 조작의 성공이나 실패를 나타내는 기발설정을 포함할수 있다.

입출력조작은 상대적으로 오랜 시간에 걸쳐 완료되며 입출력프로그람은 그 조작이 완료될 때까지 기다리게 되므로 사용자프로그람은 WRITE호출시점에서 상당히 오랜 시간동안 정지된다.

새치기와 명령주기

입출력조작이 진행되는 동안 처리기는 다른 명령집행에 사용될수 있다. 그림 1-5 ㄴ에서의 흐름조종을 고찰하자. 이전과 같이 사용자프로그람이 WRITE호출로 체계호출을 해야 할 위치에 온다. 이 경우 요구되는 입출력프로그람은 단지 코드준비와 실제적인 입출력지령으로 구성된다. 이 명령을 집행한후에 조종은 사용자프로그람에로 돌아 간다. 그 사이 외부장치는 콤퓨터기억기에서 자료를 받고 그것을 인쇄한다. 입출력조작은 사용자프로그람에서의 명령집행과 병행하여 처리된다.

외부장치가 봉사받을 준비가 되면 즉 처리기로부터 자료를 더 받을 준비가 되면 외부장치용입출력모듈은 처리기에 새치기요청신호를 보낸다. 처리기는 현행프로그람조작을 중지하고 특정한 입출력장치를 봉사하는 프로그람(새치기조종기라고 한다)에로 이행하며 그 장치를 봉사한후에 본래의 집행을 계속한다. 그림 1-5ㄴ에서는 이러한 새치기들이 발생하는 시점들을 ×로 표시하였다.

사용자프로그람의 견지에서 새치기는 바로 정상집행순서에 대한 새치기이다. 새치기처리가 끝나면 실행은 다시 회복된다(그림 1-6). 이렇게 함으로써 사용자프로그람은 새치기들을 위한 어떤 특정한 부호를 포함하지 않아도 된다. 사용자프로그람을 중지시키고 같은 시점에서 그것을 다시 회복하는것은 처리기와 조작체계가 한다.

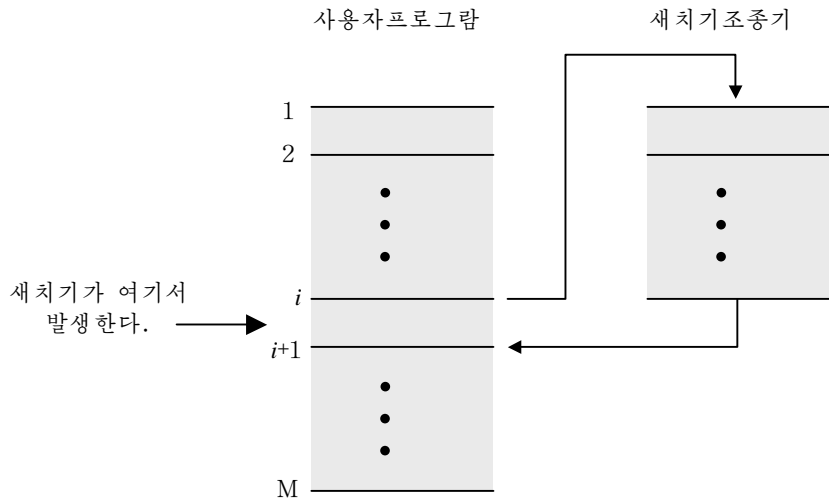
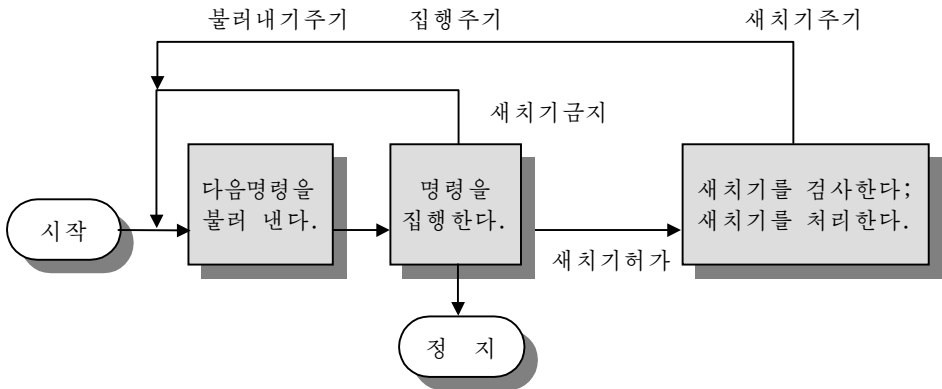



그림 1-6. 새치기를 경유하는 조종전달

새치기를 위해 그림 1-7에 보여 준것처럼 명령주기에 새치기주기가 추가된다(그림 1-2와 비교하여 보시오.). 새치기주기에서 처리기는 어떤 새치기가 발생하였는가를 검사하여 새치기신호의 유무를 가리킨다. 아무런 새치기도 발생하지 않았다면 처리기는 그 명령주기를 계속하며 현재프로그람의 다음명령을 불러 낸다. 만일 새치기가 일어 나면 처리기는 현재프로그람을 중지하고 새치기루틴을 집행한다. 새치기조종기프로그람은 일반적으로 조작체계의 일부분으로 되여 있다. 대체로 이 프로그람은 새치기특징을 판정하고 필요한 동작을 수행한다. 례를 들면 우리가 든 실례에서 조종기는 어느 입출력모듈이

그림 1-7. 새치기와 새치기주기

새치기를 발생시켰는가를 결정하고 그 입출력장치에 자료를 더 써 넣을 프로그람에로 이행할수 있다. 새치기조종기루틴이 끝나면 처리기는 새치기시점에 있는 사용자프로그람의 집행을 회복할수 있다.

명백한것은 이 처리에서 일정한 중복이 있다는것이다. 새치기의 특징을 판정하기 위하여 그리고 알맞는 동작을 결정하기 위하여 추가적인 명령들을 집행해야 한다(새치기조종기에서). 그럼에도 불구하고 입출력조작에 상대적으로 많은 시간을 랑비하게 되므로 새치기를 사용하여 처리기를 훨씬 더 효과적으로 사용할수 있다.

효률에서의 리득을 확인하기 위하여 그림 1-8을 고찰하여 보자. 이 그림은 그림 1-5 ㄱ와 1-5 ㄴ에 있는 조종흐름에 대한 시간선도이다. 그림 1-5와 1-8은 입출력조작에 요구되는 시간이 상대적으로 짧다고 즉 사용자프로그람에서 쓰기조작들사이에 있는 명령집행을 끝내는 시간보다 짧다고 가정하고 있다. 보다 일반적인 경우 특히 인쇄기와 같이 속도가 느린 장치들에서 입출력조작은 사용자명령렬을 집행하는것보다 훨씬 더 많은 시간이 든다. 그림 1-5 ㄷ는 이런 상태에 있는 사건들을 보여 준다. 이 경우 사용자프로그람은 입출력조작이 첫 호출을 끝냈다는것을 알려 주기전에 두번째 WRITE호출을 처리할수 있고 또 새로운 입출력조작을 기동할수 있다. 그림 1-9는 새치기가 있는 경우와 없는 경우에 대한 시간선도를 보여 준다. 입출력조작기간 일부 시간이 사용자명령들의 집행과 내적으로 겹쳐 지기때문에 아직 효률에서 현저한 차이가 있다는것은 잘 알수 없다.

그림 1-8. 프로그람시간선도: 짧은 입출력대기
ㄱ-새치기가 없는 경우, ㄴ-새치기가 있는 경우

새치기처리

새치기발생은 처리기장치에서 그리고 프로그람에서 몇가지 사건을 일으킨다. 그림 1-10은 대표적인 순차를 보여 준다. 입출력장치가 입출력조작을 끝내면 다음순서의 장치사건들이 발생한다.

1. 장치는 처리기에 새치기신호를 내보낸다.
2. 처리기는 새치기에 응답하기전에 현재명령집행을 완료한다(그림 1-7에 보여 준것과 같다.).
3. 처리기는 새치기를 검사하고 새치기가 있다는것을 판정하며 새치기를 내보낸 장치에 응답신호를 보낸다. 이 응답은 장치가 자기의 새치기신호를 철수하도록 한다.
4. 처리기는 조종을 새치기루틴에 넘길 준비를 하도록 요구한다. 처음 새치기시점에 있는 현재프로그람을 회복하는데 필요한 정보를 보관하도록 요구한다. 요구되는 최소정보는 프로그람상태단어 PSW와 집행해야 할 다음번 명령의 주소인 프로그람계수기의 내용이다. 이것을 체계조종탄창에 밀어 넣을수 있다(부록 1-ㄴ를 보시오.).

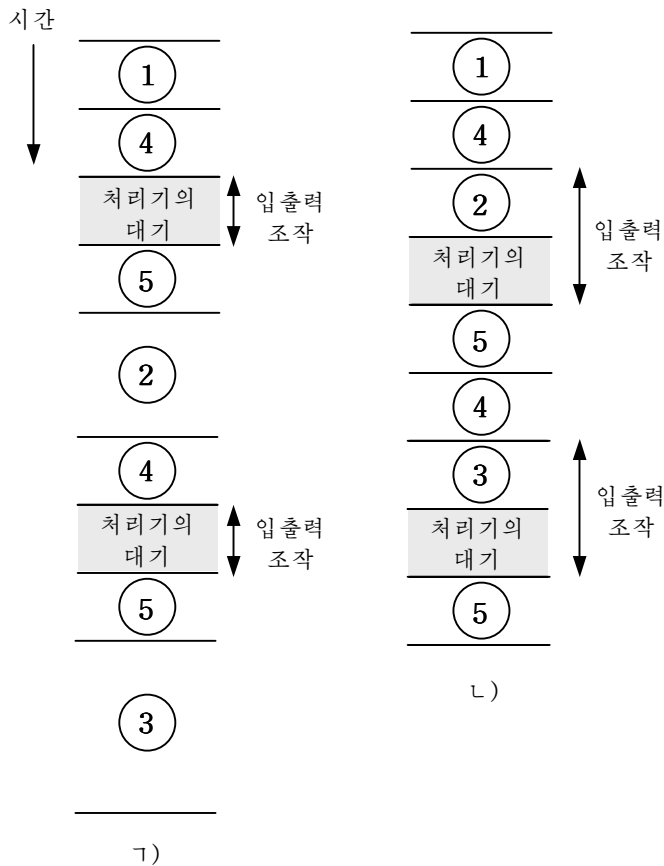



그림 1-9. 프로그람시간선도: 긴 입출력대기
ㄱ-새치기가 없는 경우, ㄴ-새치기가 있는 경우

5. 처리기는 새치기에 응답할 새치기조종프로그람의 입구주소를 프로그람계수기에 적재한다. 콤퓨터구성과 조작체계설계에 따라 매개 새치기형에 대한 프로그람 또는 매개 장치와 매개 새치기형에 대한 프로그람으로서 단일한 프로그람이 있을수 있다. 하나이상의 새치기조종루틴이 있다면 처리기는 어느것을 사용하겠는가를 판정하여야 한다. 이 정보는 발생한 새치기신호에 포함되여 있을수도 있고 요구되는 정보를 얻기 위하여 새치기를 내보낸 장치에 요청을 내보내야 할수도 있다.

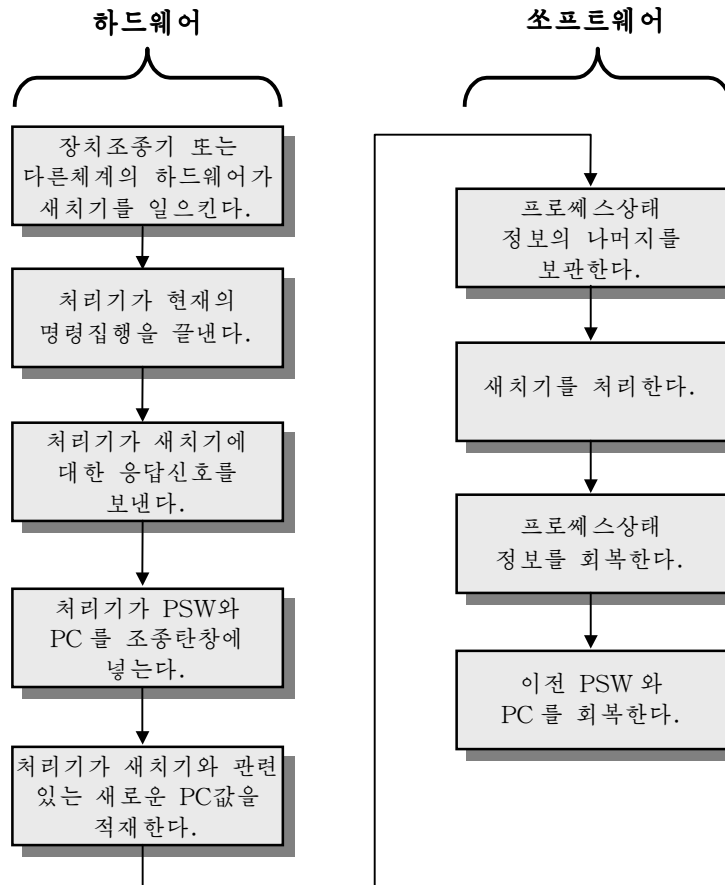



그림 1-10. 단순한 새치기처리

일단 프로그람계수기가 적재되였으면 처리기는 명령불러내기를 시작한 다음 명령주기를 계속한다. 프로그람계수기의 내용에 의하여 명령불러내기가 결정되므로 결과적으로 새치기조종기프로그람에 조종이 이송되는것으로 된다. 이 프로그람의 집행은 다음과 같은 조작들로 이루어 진다.

6. 이 시점에서 프로그람계수기와 새치기 받은 프로그람과 관계되는 PSW는 체계탄창에 보관된다. 그러나 집행프로그람의 일부 상태라고 볼수 있는 다른 정보가 있다. 특히 처리기등록기들의 내용을 보관할 필요가 있다. 그것은 새치기조종기가 이 등록기들을 사용할수 있기때문이다. 그러므로 이 모든 값들과 임의의 다른 상태정보들도 보관해야 한다. 대체로 새치기조종기는 모든 등록기들의 내용을 탄창에 보관하고 처리를 시작한다. 보관해야 할 다른 상태정보는 제3장에서 고찰한다.

그림 1-11 ㄱ는 간단한 실례를 보여 주고 있다. 이 경우에 사용자프로그람은 주소 N에 있는 명령다음에 새치기를 받는다. 모든 등록기들의 내용과 다음 명령 (N+1)의 주소는 합하여 모두 M개 단어인데 조종탄창에 들어 간다. 탄창지시기는 탄창의 새로운 꼭대기를 가리키도록 갱신되고 프로그람계수기는 새치기봉사루틴의 시작위치를 가리키도록 갱신된다.

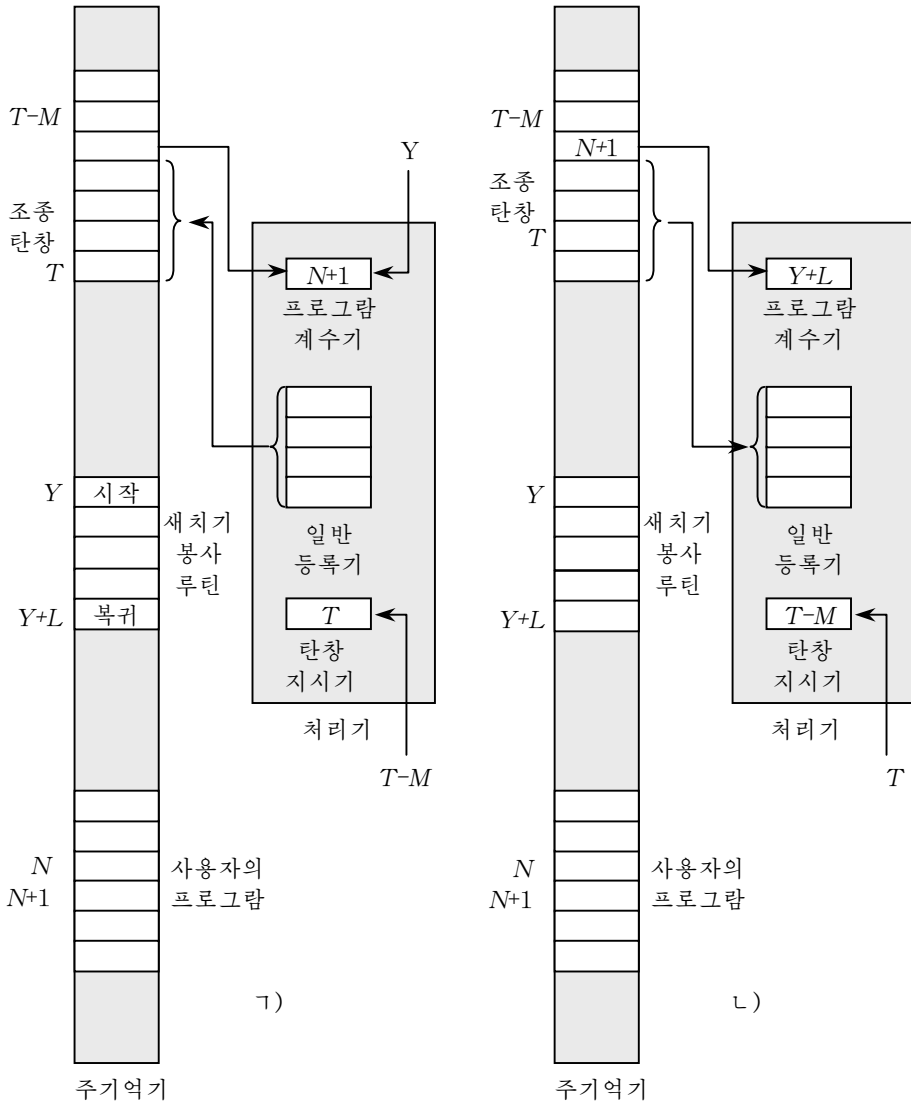



그림 1-11. 새치기로 인한 기억기와 등록기들에서의 변화
ㄱ-새치기가 주소 N에 있는 명령다음에 발생한다, ㄴ-새치기로부터 복귀

7. 새치기조종기는 이제부터 새치기를 처리해 나갈수 있다. 이것은 입출력조작이나 새치기를 일으킨 다른 사건과 관계되는 상태정보에 대한 조사를 포함하고 있다. 또한 추가적인 지령이나 응답을 입출력장치에 보낼수 있다.
8. 새치기처리가 끝나면 보관된 등록기값들이 탄창에서 나와 등록기들에 다시 넘어간다(실례 그림 1-11 ㄴ를 보시오.).

9. 최종동작은 탄창으로부터 PSW와 프로그람계수기값들을 원래대로 회복하는것이다. 결국 집행해야 할 다음명령은 이미 새치기 받은 프로그람으로부터 나오게 된다.

중요한것은 새치기 받은 프로그람에 대하여 후에 회복할수 있도록 상태정보를 모두 보관하는것이다. 이것은 새치기가 프로그람으로부터 호출된 루틴이 아니기때문이다. 물론 새치기는 임의의 순간에 그리고 사용자프로그람집행도중 임의의 위치에서 발생할수 있다. 새치기발생은 예측할수 없다.

다중새치기

지금까지는 단일새치기의 발생에 대해서만 설명하였다. 그러나 다중새치기가 발생할수 있다고 가정하자. 실례로 어떤 프로그람이 통신회선으로부터 자료를 받으면서 결과들을 인쇄할수 있다. 인쇄기는 인쇄조작을 완료할 때마다 새치기를 발생시킨다. 통신회선은 한조의 자료가 도착할 때마다 새치기를 발생시킨다. 그 자료는 한개 문자일수도 있고 한개 블로크일수도 있는바 그것은 통신능력에 관계된다. 어느 경우이든 인쇄기를 처리하고 있는 기간에 통신새치기가 발생할수 있다.

다중새치기는 두가지 방법으로 취급할수 있다. 첫째로는 어떤 새치기를 처리하고 있는 동안에는 다른 새치기들을 허용하지 않는 방법이다. 금지형새치기는 단순히 처리기가 임의의 새로운 새치기요청신호를 무시할수 있으며 무시하게 된다는것을 의미한다. 이 시간내에 발생하는 새치기는 미정으로 되며 처리기가 새치기들을 다시 허용한후에 처리기가 그것을 검사하게 된다. 이와 같이 사용자프로그람이 집행중에 있을 때 새치기가 발생하면 그것은 즉시에 무효로 된다. 새치기조종기루틴이 끝난다음 사용자프로그람을 회복하기전에 새치기들은 다시 허용되며 처리기는 추가적인 새치기들이 발생했는가를 검사한다. 이 방법은 새치기들을 엄밀한 순서로 조종하므로 단순하고 간단하다(그림 1-12 ㄱ).

이 방법의 약점은 상대적인 우선권과 시간림계요구를 고려하지 않는다는것이다. 실례로 입력이 통신회선으로부터 들어 올 때 입력을 빨리 처리하여 입력을 더 받기 위한 공간을 만들어야 한다. 만일 두번째 묶음이 들어 오기전에 첫 입력묶음을 처리하지 못한다면 입출력장치의 완충기가 차고 넘칠수 있으므로 자료를 잃어 버릴수 있다.

둘째 방법은 새치기들에 우선권을 정해 놓고 높은 우선권을 가진 새치기가 낮은 우선권을 가진 새치기조종기를 새치기하도록 하는것이다(그림 1-12 ㄴ). 둘째 방법에 대한 실례로 인쇄기, 디스크와 통신회선으로 구성된 세개의 입출력장치를 가진 체계를 고찰하여 보자. 우선권은 각각 2, 4, 5순서로 놓는다고 하자. 그림 1-13은 가능한 순서를 보여준다. 사용자프로그람은 t=0에서 시작한다. t=10에서 인쇄기새치기가 발생하고 사용자정보는 체계탄창에 들어 가며 집행은 인쇄기새치기봉사루틴(ISR)에서 계속한다. 이 루틴이 아직 집행되고 있는중인 t=15에서 통신새치기가 발생한다. 통신회선이 인쇄기보다 높은 우선권을 가지고 있기때문에 새치기가 허용된다. 인쇄기 ISR는 새치기되며 그 상태는 탄창에 들어 가고 집행은 통신 ISR에서 계속된다. 이 루틴을 집행하고 있는중에 디스크새치기가 발생한다(t=20). 이 새치기는 보다 낮은 우선권을 가지고 있기때문에 보류되고 통신 ISR가 집행을 완료한다.

통신 ISR가 끝나면(t=25) 이전 처리기상태가 회복된다. 이전 처리기상태는 인쇄기 ISR의 집행이다. 그러나 그 루틴에서 단 한개의 명령도 집행될 가능성을 주지 않고 처리기는 보다 높은 우선권을 가진 새치기를 허용하며 조종은 디스크 ISR에 이송된다. 루틴이 완료될 때(t=35)에야 비로소 인쇄기 ISR가 회복된다. 루틴이 완료되였을 때(t=40) 조종은 사용자프로그람에로 돌아 간다.

ㄴ)

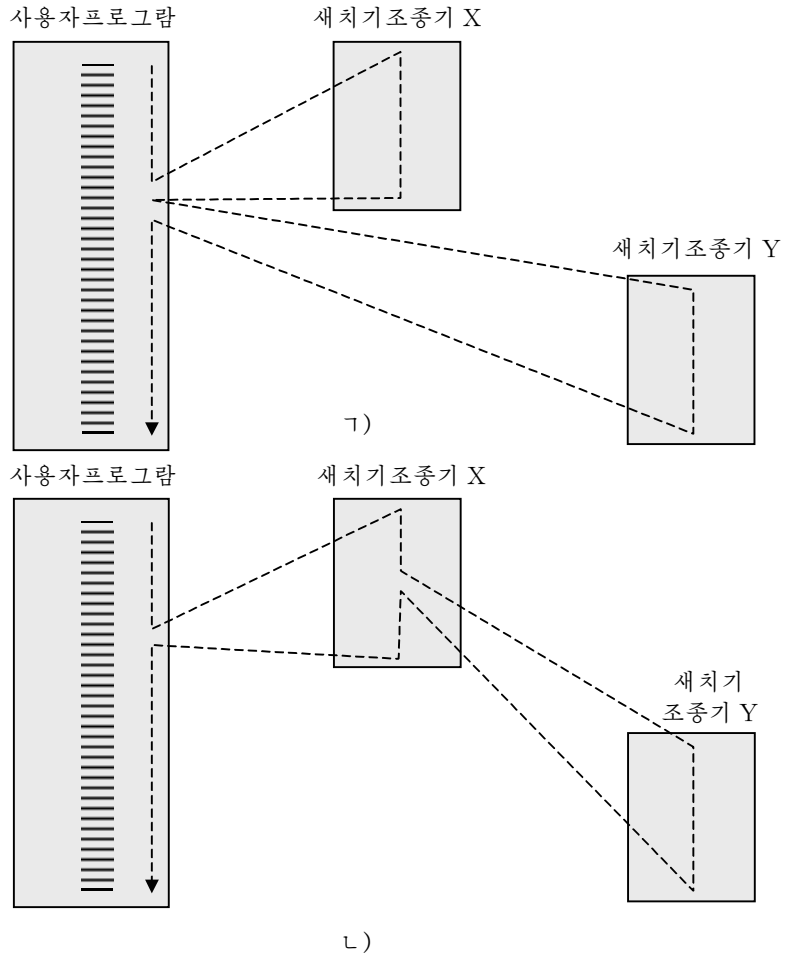



그림 1-12. 다중새치기에 의한 조종의 이송
ㄱ-순차적인 새치기처리, ㄴ-겹친 새치기처리

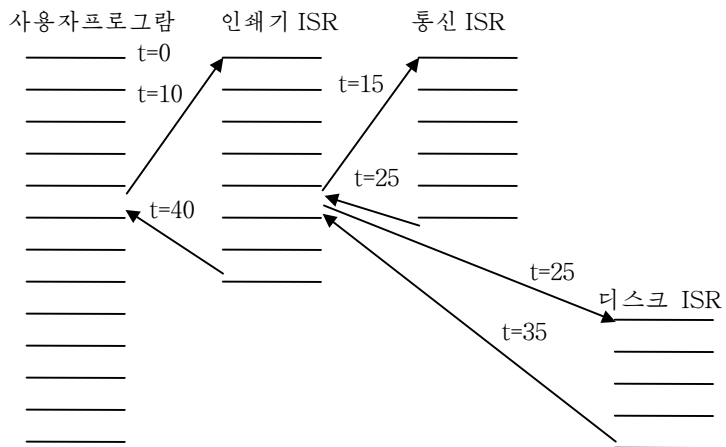



그림 1-13. 다중새치기의 시간순서실례[TANE90]

다중프로그람처리

새치기들을 사용한다고 하여도 처리기를 완전히 효률적으로 쓸수 없다. 실례로 그림 1-9 ㄴ는 처리기를 더 잘 사용할수 있다는것을 보여 주고 있다. 입출력조작을 완료하는데 요구되는 시간이 입출력호출들사이의 사용자코드보다 훨씬 더 크다면(일반적인 상태이다.) 처리기는 많은 시간 정지하게 된다. 이 문제에 대한 해결방도는 다중사용자프로그람을 같은 시간에 동작하도록 하는것이다.

실례로 처리기가 두개의 프로그람을 집행해야 한다고 가정하자. 하나는 기억기로부터 자료를 읽어 외부장치에 내보내는것이고 다른 하나는 많은 계산을 해야 하는 응용프로그람이라고 하자. 처리기는 출력프로그람을 시작하여 외부장치에 쓰기지령을 내보낼수 있으며 그다음 응용프로그람을 집행할수 있다. 처리기가 많은 프로그람들을 취급할 때 어느 프로그람을 집행하겠는가 하는 순서는 그것들의 상대적인 우선권은 물론 입출력을 기다리고 있는가 아닌가 하는데 관계될것이다. 프로그람이 새치기되여 조종이 새치기조종기에 이송되였다면 일단 새치기조종루틴은 완료되고 조종은 그때 집행중에 있던 사용자프로그람에 즉시로 되돌아 가지 않을수도 있다. 그대신 조종은 보다 높은 우선권을 가진 다른 프로그람에로 건너 뛸수 있다. 결국 새치기된 사용자프로그람은 자기가 제일 높은 우선권을 가졌을 때 회복된다. 집행상순서를 가지는 다중프로그람들에 대한 개념을 다중프로그람처리라고 하며 제2장에서 더 설명한다.

제 5 절. 기억기의 계층구조

콤퓨터의 기억기에 대한 설계제약조건은 세가지 질문으로 요약할수 있다. 즉 얼마만한 량으로? 얼마나 빨리? 얼마만한 가격으로?

량적크기에 대한 문제는 어느 정도 한계가 있다. 용량이 주어 지면 응용에서는 그것을 사용하도록 개발할수 있다. 속도상문제는 의미상 대답하기 더 쉽다. 가장 빠른 동작을 달성하자면 기억기가 CPU와 보조를 맞출수 있어야 한다. 즉 처리기가 명령들을 집행함에 있어서 명령이나 연산수들을 기다리면서 정지하는 일이 없도록 해야 한다. 최종문제도 고찰해 보아야 한다. 실지 체계들에서 기억기의 비용은 다른 구성성분들과의 관계로 볼 때 무시할수 없다.

예측할수 있는것은 기억기에 대한 세가지 주요특성 즉 비용, 용량 및 호출시간사이에 절충방안이 있다는것이다. 주어 진 시대에 기억기체계를 실현하기 위하여 각이한 방안을 사용하게 된다. 이 여러가지 방안들속에서 다음의 관계가 보존된다.

- 접근시간이 빠를수록 비트당 비용은 더 커 진다.
- 용량이 클수록 비트당 비용은 더 작아 진다.
- 용량이 클수록 접근시간은 떠 진다.

설계자들이 고충을 겪는것은 명백하다. 설계자는 큰 용량의 기억기를 제공하는 방안을 쓰려고 한다. 그것은 큰 용량이 필요하며 또 비트당 비용도 낮기때문이다. 그러나 동작요구를 만족시키기 위해 설계자는 값이 비싸지만 접근시간은 빠르고 용량이 작은 기억기를 쓰려고 한다.

이것을 해결하기 위한 방도는 단일한 기억기요소나 기술에 의거할것이 아니라 기억기계층구조를 사용하는것이다. 대표적인 계층구조를 그림 1-14에 보여 주었다. 분층은 다음과 같은 특성이 있다.

① 비트당 비용이 작아 진다.
② 용량이 커진다.
③ 접근시간이 증가한다.
④ 처리기가 기억기에 접근하는 빈도는 작아 진다.

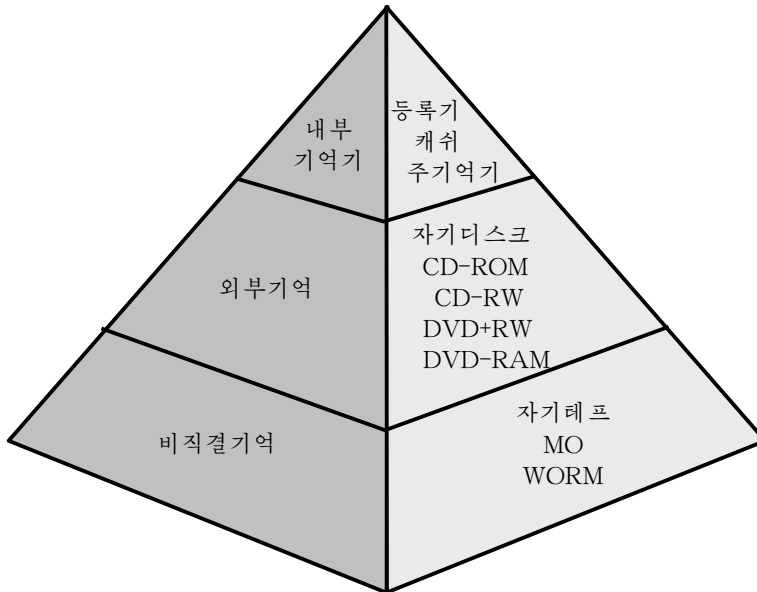



그림 1-14. 기억기의 계층구조

이렇게 함으로써 보다 작고, 값이 비싸며, 속도가 빠른 기억기들은 보다 크고, 눅으며, 속도가 느린 기억기들로 보충된다. 이 구성방식을 성공시킬수 있는 열쇠는 마지막항목 즉 접근빈도를 줄이는데 있다. 캐쉬를 설명할 때 그리고 가상기억기층을 설명할 때 이 개념을 더 구체적으로 론의하기로 하자. 기본적인 설명은 여기서 한다.

처리기가 2준위의 기억기에 접근한다고 하자. 1준위는 1000개의 단어를 가지고 있으며 접근시간은 0.1㎲이다. 2준위는 100,000단어를 가지고 있으며 접근시간은 1㎲이다. 접근하여야 할 단어가 1준위에 있으면 처리기는 그것에 직접 접근하며 2준위에 있으면 우선 그 단어를 1준위에로 전송하고 그다음 접근한다고 가정하자. 간단히 하기 위하여 처리기가 그 단어가 1준위에 있는가 또는 2준위에 있는가를 판정하는데 걸리는 시간은 무시한다. 그림 1-15은 이 상태를 포괄하는 일반적인 곡선모양을 보여 주고 있다. 그림은 2준위기억기에 대한 평균접근시간을 **명중률** H의 함수로 보여 준다. 여기서 H는 고속기억구조(실례로 캐쉬)에서 차지하는 모든 기억기접근의 몫으로 정의된다. T_1 은 1준위에 대한 접근시간이며 T_2 는 2준위에 대한 접근시간이다.[3] 1준위에 대한 접근이 많아 질수록 총적인 평균접근시간은 2준위보다 1준위접근시간에 훨씬 더 가깝다는것을 알수 있다.

우리가 든 실례에서 95%의 기억기접근이 캐쉬에서 발견된다고 가정하자(H=0.95). 그러면 단어에 접근하는 평균시간은 다음과 같이 표시할수 있다.

$$(0.95)(0.1\mu s)+(0.05)(0.1\mu s+1\mu s)=0.095+0.055=0.15\mu s$$

[3] 접근된 단어가 고속기억기에서 발견되면 그것을 **명중**으로 정의한다. 접근된 단어가 고속기억기에서 발견되지 않으면 **실패**가 일어 난다.

결과는 고속기억기의 접근시간에 아주 가깝다. 중요한것은 원리적인 해결대책을 세워야 하며 그것은 조건 ①~④를 적용함으로써만 할수 있다. 어떤 방안을 적용하든 조건 ①~③을 만족시키는 기억기체계는 무수히 많다. 조건 ④도 일반적으로 유효하다.

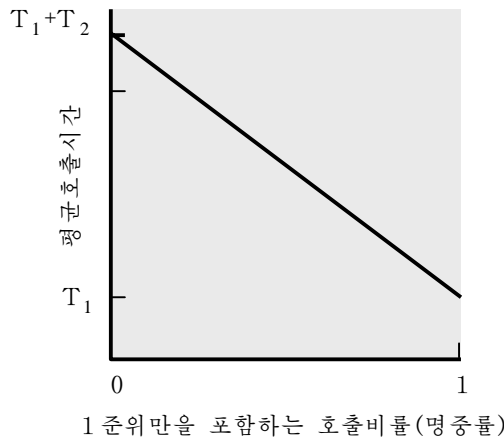



그림 1-15. 단순한 2준위기억기의 동작(명중률)

조건 ④의 유효성에 대한 기초는 참조의 국소성이라고 하는 원리이다[DENN 68]. 프로그람을 집행하는 과정에 명령과 자료를 위해 처리기가 기억기를 참조하는것은 클라스터를 짓는 경향성을 가진다. 프로그람들은 대체로 반복고리들과 보조루틴들을 가지고 있다. 일단 고리나 보조루틴에 들어 가면 작은 명령묶음에 대한 참조가 반복된다. 류사하게 표와 배렬들에 대한 조작은 클라스터를 이루는 자료단어모임을 호출하게 된다. 오랜 기간에는 클라스터가 사용중에 변하지만 짧은 기간에는 처리기가 초보적으로 고정된 클라스터형기억기를 참조하면서 동작한다.

따라서 차례로 낮아 지는 개개의 준위에 대한 접근비률이 웃준위에 대한 접근보다 본질적으로 작은 계층구조에서 자료를 조작할수 있다. 이미 언급한 2준위실례를 고찰하자. 2준위기억기가 모든 프로그람명령과 자료를 가지고 있다고 하자. 현재클라스터는 일시적으로 1준위에 놓여 있을수 있다. 시간이 지나감에 따라 1준위에 있는 클라스터들중의 하나를 2준위의것과 바꾸어 1준위에로 들어 오는 새 클라스터를 위한 공간을 만들어야 한다. 그러나 평균적으로 대부분의 참조과정은 1준위에 있는 명령과 자료에서 일어 난다.

이 원리를 2준위이상의 기억기에 적용할수 있다. 가장 빠르고 가장 작으며 가장 값비싼형의 기억기는 처리기안의 등록기들로 이루어 져 있다. 일부 기계들은 수백개의 등록기를 가지고 있지만 대체로 처리기는 몇타스의 등록기를 가질것이다. 2준위에로 되돌아 간다면 주기억기는 콤퓨터의 원리적인 내부기억체계이다. 주기억기에서 매개 위치는 유일한 주소를 가지며 대부분의 기계명령들은 하나이상의 주기억기주소들을 참조한다. 주기억기는 보통 속도가 빠르고 하나이상의 주기억기들을 참조한다. 주기억기는 보통 속도가 빠르고 작은 캐쉬로 확장된다. 캐쉬는 보통 프로그람작성자가 변경시킬수 없으며 실제상 처리기도 변경시킬수 없다. 그것은 주기억기와 처리기등록기들사이에서 동작을 개선하기 위하여 자료의 이동을 계획화하여 주는 장치이다.

방금 설명한 세가지 형태의 기억기는 대체로 휘발성이며 반도체기술을 사용한다. 세개의 준위를 사용하는것은 속도와 비용에서 차이가 있는 반도체기억기들이 여러가지 형태로 존재한다는 사실을 받아 들였기때문이다. 자료는 외부의 큰 기억장치들에 보다 영

구적으로 기억된다. 그중에서 가장 일반적인것은 하드디스크이며 지울수 있는 디스크, 테프 및 빛기억장치와 같이 지울수 있는 매체이다. 외부에 있는 비휘발성기억기를 **2차기억기** 또는 **보조기억기**라고도 한다. 이것들은 프로그람과 자료파일들을 기억시키는데 쓰이며 개별적인 바이트나 단어들과 달리 보통 파일과 레코드에 의해서만 프로그람작성자에게 보인다. 디스크는 또한 가상기억기로 알려 져 있는 주기억기에로 확장하는데도 쓰인다. 제8장에서 이 내용을 설명한다.

보충적인 준위들을 쏘프트웨어적으로 계층구조에 효과적으로 추가할수 있다. 실례로 주기억기의 일부분을 디스크로 읽어 내보내는 자료를 일시적으로 유지하기 위한 완충기로 사용할수 있다. 이러한 수법을 때때로 디스크캐쉬라고 한다(제11장에서 구체적으로 설명한다.). 디스크캐쉬는 두가지 방법으로 동작을 개선한다.

- 디스크쓰기가 클라스터를 이룬다. 작은 자료이송을 많이 하는 대신에 몇개의 큰 자료이송을 한다. 이것은 디스크동작을 개선시키며 처리기포함을 최소화한다.
- 디스크에 다음번 덤프하기전에 써내기용으로 지정된 일부 자료를 프로그람이 참조할수 있다. 이 경우에 자료는 느린 디스크로부터가 아니라 쏘프트웨어캐쉬로부터 빨리 찾아 볼수 있다.

부록 1-ㄱ는 다준위기억기구조의 동작꾸밈을 론의한다.

제 6 절. 캐쉬

캐쉬는 조작체계가 변경시킬수 없지만 다른 기억기관리장치와 대화한다. 더우기 가상기억기구조에 사용된 많은 원리들은(제8장에서 설명한다.) 캐쉬에도 적용된다.

캐쉬의 필요성

모든 명령주기에서 처리기는 적어도 한번 기억기에 접근하여 명령을 불러 내며 흔히 추가적으로 여러번 접근하여 연산수들을 불러 내거나 결과를 기억시킨다. 처리기가 명령을 집행할수 있는 시간은 명백히 기억주기시간으로 인한 제한을 받는다. 처리기와 기억기속도사이의 지속되는 불일치로 하여 이 제한은 더 빨리 증가하고 있다. 부닥치는 문제는 속도, 비용 및 용량크기사이의 절충방안에 대한것이다. 리상적으로는 주기억기를 처리기등록기와 같은 기술로 제작하여 기억기주기시간을 처리기주기시간과 비교할수 있을만큼 만들어야 한다. 그런데 이것은 너무 값 비싼 방법이다. 해결방도는 처리기와 주기억기사이에서 작고도 빠른 기억기 즉 캐쉬를 넣어 줌으로써 국소성의 원리를 받아 들이는데 있다.

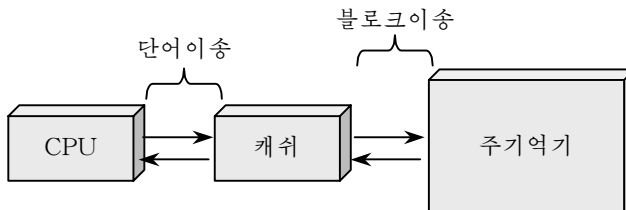



그림 1-16. 캐쉬와 주기억기

캐쉬의 원리

캐쉬는 기억기속도를 가능한한 가장 빠른 기억기속도에 접근시키며 동시에 값 비싼 반도체기억기들보다 적은 비용으로 큰 기억용량을 보장할것을 지향하고 있다. 이 개념을 그림 1-16에서 설명하고 있다. 상대적으로 크고 느린 주기억기와 함께 보다 작고 고속인 캐쉬가 있다. 캐쉬는 주기억기의 일부분에 대한 복사를 포함하고 있다. 기억기의 단어를 읽으려고 할 때 그 단어가 캐쉬에 있는가를 판정하기 위한 검사를 진행한다. 만일 있다면 그 단어는 처리기에 들어 간다. 만일 없으면 일정하게 고정된 수의 단어로 이루어 지는 주기억기블로크를 캐쉬에 읽어 들이고 다음에 그 단어가 처리기에 들어 간다. 참조의 국소성현상으로 인하여 자료블로크가 캐쉬에 들어 가면 그 블로크내에 있는 다른 단어를 후에 참고할수 있으므로 단일기억기참조를 만족시킬수 있다.

그림 1-17은 캐쉬/주기억기체계의 구조를 보여 주고 있다. 주기억기는 2^n개의 주소화할수 있는 단어로 구성되며 매개 단어는 단일한 *n*bit주소를 가지고 있다. 주소사영의 목적으로부터 이 기억기가 각각 K개 단어로 된 많은 고정길이블로크로 되여 있다고 보자. 즉 $M=2^n/K$개의 블로크가 있다. 캐쉬는 각각 K개 단어를 가지는 C개의 홈으로 구성되여 있고 그 홈의 수는 주기억기블로크의 수보다는 상당히 작다(C<<M).[4] 주기억기의 일정한 블로크들의 부분모임이 캐쉬내의 홈들에 있다. 캐쉬에 없는 기억기블로크내의 단어를 읽는다면 블로크는 캐쉬의 어느 한 홈에로 이송된다. 블로크들은 홈보다 더 많기때문에 개별적인 홈이 어떤 특정한 블로크에 유일하게 영구적으로 배당될수도 없다. 따라서 매개 홈은 특정한 블로크가 현재 기억되여 있다는것을 식별해 주는 하나의 태그를 포함하고 있다. 태그는 보통 몇개의 상위주소비트이며 비트들의 련속으로 시작하는 모든 주소를 가리킨다.

간단한 실례로 6bit의 주소와 그 비트의 태그를 가지고 있다고 하자. 태그 01은 다음의 주소를 가진 기억위치의 블로크들을 참조한다. 즉

010000, 010001, 010010, 010011, 010100, 010101, 010110, 010111, 011000, 011001, 011010, 011011, 011100, 011101, 011110, 011111

그림 1-18은 읽기조작을 설명하고 있다. 처리기는 읽으려는 단어의 주소를 발생시킨다. 만일 단어가 캐쉬에 포함된다면 그것이 처리기에 들어 간다. 그렇지 않으면 그 단어를 포함하는 블로크가 캐쉬에 적재되고 단어는 처리기에 들어 간다.

캐쉬의 설계

캐쉬설계에 대한 구체적인 설명은 이 책의 고찰범위를 벗어 난다. 여기서는 주로 기본적인 요인들을 고찰하기로 한다. 가상기억기와 디스크캐쉬설계를 취급하는데서 류사한 설계상의 문제점들을 고찰하게 된다는것을 알수 있다. 그 문제점들은 다음과 같은 범주들에 귀착된다.

- 캐쉬의 크기
- 블로크크기
- 사영기능
- 치환알고리듬
- 써넣기방책

---

4. 기호 <<는 《훨씬 더 작다.》를 의미한다. 류사하게 기호 >>는 《훨씬 더 크다.》를 의미한다.

우리는 이미 **캐쉬크기**문제를 취급하였다. 합리적이며 작은 캐쉬가 동작에 일정한 영향을 준다는데 대해서는 고찰하였다. 다른 크기문제는 바로 **블로크크기** 즉 캐쉬와 주기억기사이에서 교환되는 자료의 단위에 대한 문제이다. 블로크크기는 매우 작은 크기로부터 큰 크기에로 증가됨에 따라 국소성의 원리에 의하여 우선 명중률이 증가한다. 국소성의 원리는 참조한 단어의 근방에 있는 자료를 다음번에 참조할 가능성이 크다는것이다.

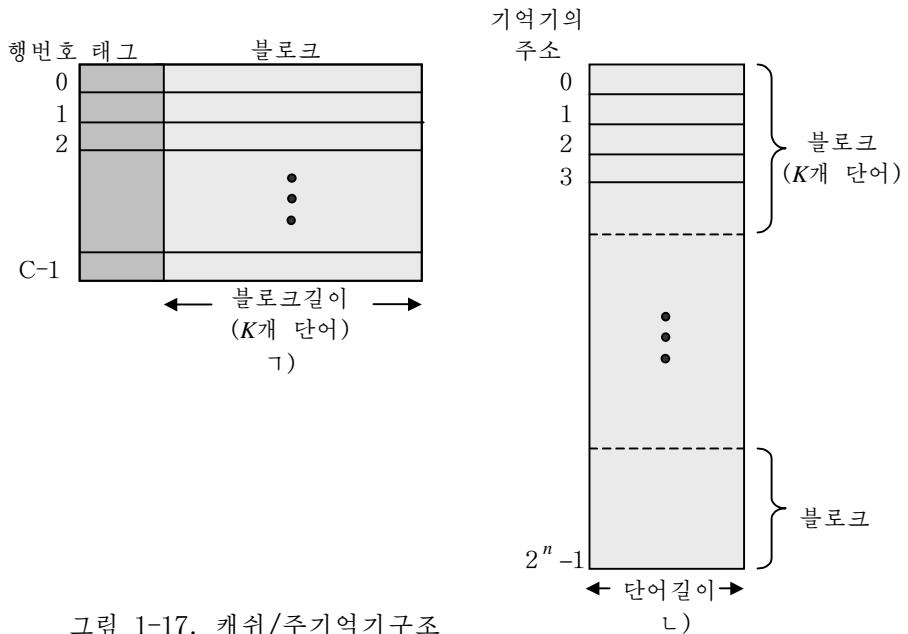



그림 1-17. 캐쉬/주기억기구조
ㄱ-캐쉬, ㄴ-주기억기

블로크크기를 증가시킴에 따라 유효한 자료가 캐쉬에 더 많이 들어 오게 된다. 그러나 좀 더 커지면 명중률은 감소하기 시작하며 새로 불러 낸 자료를 사용할 가능성은 블로크용으로 빈 공간을 만들기 위하여 캐쉬에서 옮겨야 할 자료를 다시 사용할 가능성보다 작아 지기 시작한다.

새로운 블로크의 자료를 캐쉬에 읽어 넣을 때 **주소사영기능**은 블로크가 캐쉬의 어느 위치를 차지하게 되는가를 판정한다. 두개의 제한조건이 사영기능에 영향을 준다. 첫째로, 한 블로크를 읽어 넣을 때 다른 블로크를 교체해야 한다. 곧 다시 요구될 블로크를 교체할 가능성이 최소로 되도록 진행하는것이 좋을것이다. 배치기능이 유연해지면 질수록 명중률이 최대로 되게 알고리듬을 설계할수 있는 범위는 더 커진다. 둘째로, 배치기능이 유연해질수록 주어 진 블로크가 캐쉬내에 있는가를 판정하기 위하여 캐쉬를 탐색하는데 필요한 회로는 더 복잡해 진다.

치환알고리듬은 배치기능의 제한조건내에서 새로운 블로크를 캐쉬에 넣으려고 하는데 그 캐쉬가 이미 다른 블로크들로 채워 진 모든 블로크를 가질 때 교체해야 할 블로크를 선정한다. 곧 다시 요구될 블로크를 치환하는것이 좋을것이다. 이러한 블로크를 식별한다는것은 불가능하더라도 합리적이며 효과적인 방도는 더이상 참조되지 않으면서 캐쉬에 제일 오래 동안 머물러 있는 블로크를 치환하는것이다. 방책은 최근 최소사용알고리듬(LRU)이라고 한다. 이때 최근에 최소로 사용된 블로크를 식별하기 위한 장치기구가 있어야 한다.

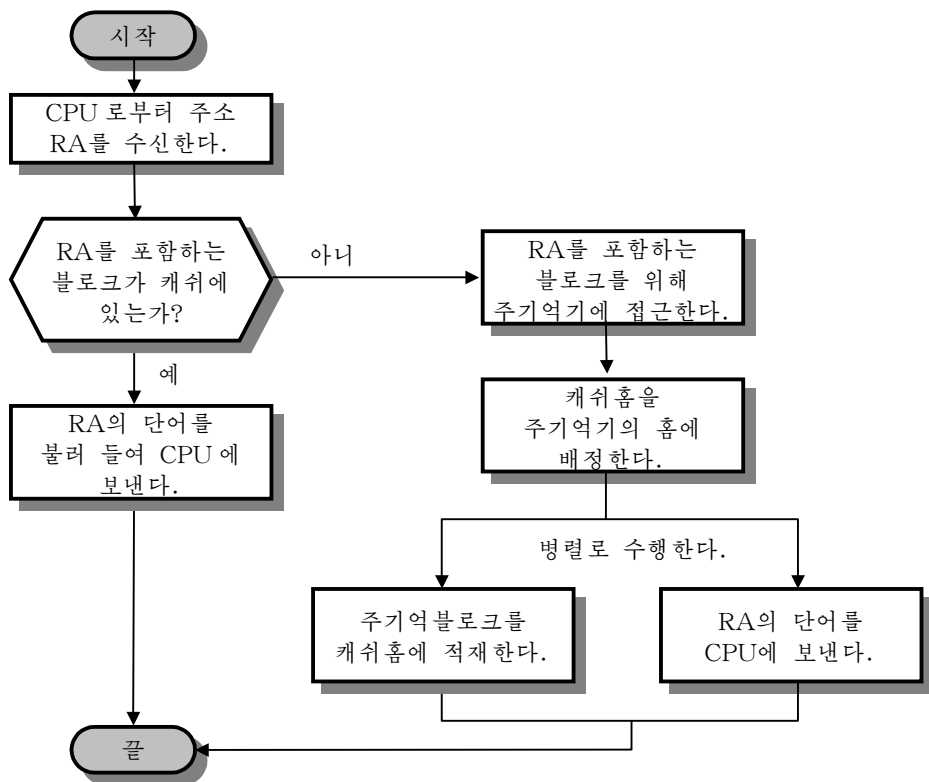



그림 1-18. 캐쉬의 읽기조작

캐쉬에 있는 블로크의 내용을 바꾼다면 그것을 치환하기전에 주기억기에 다시 써 넣어야 한다. **써넣기방책**은 언제 기억기써넣기조작이 일어 나는가를 지시한다. 한쪽 극단에서는 블로크가 갱신될 때마다 매번 써넣기를 진행할수 있다. 다른쪽 극단에서는 블로크가 교체될 때에만 써넣기를 진행할수 있다. 뒤의 방법은 기억기쓰기조작을 최소로 되게 하지만 주기억기를 낡은 상태에 머물러 있게 한다. 이것은 다중처리기조작과 입출력모듈에 의한 직접기억접근에 영향을 줄수 있다.

제 7 절. 입출력통신기술

입출력조작을 위한 세가지 수법이 가능하다.

- 프로그람식입출력
- 새치기구동식입출력
- 직접기억접근

프로그람식입출력

처리기가 프로그람을 집행하면서 입출력과 관련되는 명령을 만나면 해당한 입출력모듈에 지령을 내보내여 명령을 집행한다. 프로그람식입출력에서 입출력모듈은 요구되는 동작을 수행하고 입출력상태등록기에서 해당한 비트를 설정한다. 입출력모듈은 처리기에 대하여 더이상 아무 동작도 하지 않는다. 특히 처리기를 새치기하지 않는다. 이렇게 하여 처리기의 책임은 동작이 끝나는것을 확인할 때까지 입출력모듈의 상태를 주기적으로 검

사하는것으로 된다.

이러한 방법으로 처리기는 주기억기로부터 출구할 자료를 내보내고 주기억기에 입력할 자료를 기억시키는 책임을 진다. 입출력쏘프트웨어는 장치상태를 검출하며 읽기나 쓰기지령을 보내며 자료를 이송시키는 등의 입출력조작을 직접 조종하는 명령들을 처리기에서 집행시키도록 하는 방법으로 작성한다. 그러므로 명령모임은 다음과 같은 범주들내에 있는 입출력명령들을 가진다.

- **조종 :** 외부장치를 동작시키며 해야 할 일을 알려 주는데 쓰인다. 실례로 자기테프장치와 한개의 레코드를 후진시키든가 전진시키도록 명령을 줄수 있다.
- **상태 :** 입출력모듈과 그 주변장치들과 관련된 여러가지 상태조건을 검사하는데 쓰인다.
- **이송 :** 처리기등록기들과 외부장치들사이에서 자료를 읽거나 쓰는데 쓰인다.

그림 1-19 ㄱ는 프로그람식입출력이 외부장치로부터 자료블로크(실례로 테프로부터 한개의 레코드)를 기억기에 읽어 넣기 위한 사용실례를 보여 주고 있다. 자료는 한번에 한단어(16bit)를 읽는다. 읽어 들인 매개 단어에 대하여 처리기는 단어가 입출력모듈의 자료등록기에서 유효한가를 판정할 때까지 상태검사주기에 남겨 놓아야 한다. 이 흐름도는 이 수법의 기본부족점 즉 처리기가 필요없이 동작상태에 있게 되는 시간소비처리에 초점을 두고 있다.

새치기구동식입출력

프로그람식입출력에서 문제는 자료를 더 수신하겠는가 더 송신하겠는가 하는것과 관련하여 처리기가 입출력모듈을 오랜 시간동안 기다려야 한다는것이다. 처리기는 기다리는동안 입출력모듈의 상태를 반복하여 문의해야 한다. 결과적으로 총적인 체계의 동작준위는 심히 떨어 지게 된다.

다른 방법으로서는 처리기가 모듈에 입출력지령을 내보낸 다음 다른 유용한 일을 계속해 나가도록 하는것이다. 처리기와 자료교환할 준비가 되였을 때 입출력모듈은 처리기를 새치기하여 봉사를 요청한다. 처리기는 그때 이전과 같이 자료이송을 집행한 다음 자기의 처리를 계속한다.

우선 입출력모듈의 견지에서 입력이 어떻게 진행되는가를 고찰하자. 입출력모듈은 처리기로부터 READ지령을 받는다. 다음 입출력모듈은 관련이 있는 주변장치로부터 자료를 읽어 들인다. 일단 자료가 모듈의 등록기에 있으면 모듈은 조종회선을 통하여 처리기에 새치기신호를 보낸다. 이때 입출력모듈은 처리기가 그 자료를 요구할 때까지 기다린다. 요구가 나타나면 모듈은 자료를 자료모선상에 내보내고 다른 입출력조작을 준비한다.

처리기가 하는 입력동작은 다음과 같다. 처리기는 우선 READ지령을 내보낸다. 다음 현재프로그람의 내용(실례로 프로그람계수기와 처리기등록기들)을 보관하고 빠져 나와 그밖의 일정한 일을 한다(실례로 처리기가 같은 시간에 몇개의 서로 다른 프로그람들상에서 동작하고 있을수 있다.). 매개 명령주기의 마감에 처리기는 새치기를 검사한다(그림 1-7). 입출력모듈로부터 새치기가 발생하면 처리기는 현재 집행하고 있는 프로그람의 내용을 보관하고 새치기처리를 위해 새치기조종프로그람을 집행하기 시작한다. 이 경우에 처리기는 입출력모듈로부터 자료단어를 읽고 그것을 기억기에 기억시킨다. 그다음 입출력모듈로부터 자료단어를 읽고 그것을 기억기에 기억시킨다. 그다음 입출력지령을 내보냈던 프로그람(또는 다른 프로그람)의 내용을 회복하고 집행한다.

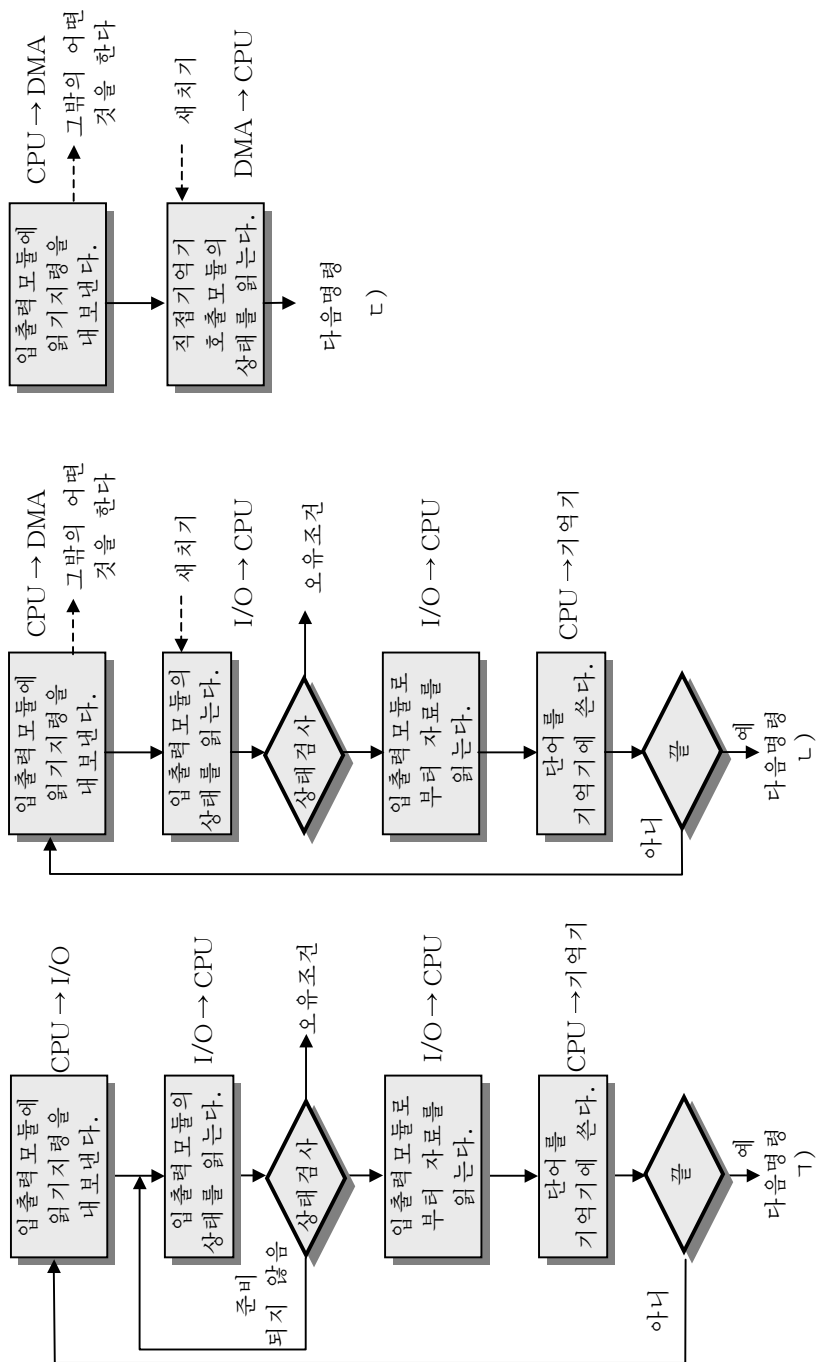



그림 1-19. 자료블로크입력에서의 세가지 수법
ㄱ-프로그람식입출력, ㄴ-새치기구동식입출력, ㄷ-직접기억접근

그림 1-19 ㄴ는 새치기구동식입출력을 사용하여 자료블로크를 읽어 들이는 과정을 보여 주고 있다. 그림 1-19 ㄱ와 비교해 보자. 새치기구동식입출력은 불필요한 대기시간을 없애 버리기때문에 프로그람식입출력보다 더 효과적이다. 그러나 새치기구동식입출력은 아직 많은 처리기시간을 소비한다. 그것은 기억기로부터 입출력모듈로 또는 입출력모듈로부터 기억기에로 가는 매 자료단어가 처리기를 통과해야 하기때문이다.

콤퓨터체계에는 거의나 다중입출력모듈들이 있기때문에 어느 장치가 새치기를 일으켰는가를 처리기가 판정할수 있도록 그리고 다중새치기인 경우 어느것을 우선 조종하겠는가를 결심할수 있도록 해주는 기구가 있어야 한다. 일부 갖추어 진 체계들에는 다중새치기들이 있어서 매개 입출력모듈이 서로 다른 선상에 신호를 보낸다. 매개 회선은 서로 다른 우선권을 가진다. 다른 방법으로서 단일한 새치기선이 있을수 있지만 장치주소를 유지하기 위하여 추가적인 선들을 사용하여야 한다. 또한 서로 다른 장치들에도 각이한 우선권이 할당된다.

직접기억접근

새치기구동식입출력은 단순한 프로그람식입출력보다 효과적이지만 기억기와 입출력모듈사이에서 처리기가 자료를 이송시키는데 더 적극적으로 간섭할것을 요구하며 처리기를 거치지 않고 임의의 자료를 이행시킬것을 요구하고 있다. 때문에 입출력형태들은 모두 두개의 내부약점을 가지고 있다.

1. 입출력이송속도는 처리기가 장치를 검사하고 봉사할수 있는 속도로 인한 한계를 가진다.
2. 처리기는 입출력이송을 관리하는데 매이게 된다. 즉 매개 입출력이송을 위하여 매개의 명령을 집행해야 한다.

큰 용량의 자료를 옮길 때보다 효과적인 기능 즉 직접기억접근(DMA)기능이 요구된다. DMA기능은 체계모선상에서 독립적인 모듈로 수행할수도 있고 입출력모듈에 포함시킬수도 있다. 어느 경우나 DMA는 다음과 같이 동작한다. 처리기가 자료블로크를 읽거나 쓰려고 할 때에는 DMA모듈에 지령을 내보낸다. DMA모듈에 보내는 지령은 다음과 같은 정보를 포함하고 있다.

- 읽기 또는 쓰기의 요청
- 입출력장치의 주소
- 기억기에서 읽거나 써넣어야 할 시작위치
- 읽거나 쓰려는 단어의 수

처리기는 그다음 다른 일을 계속한다. 처리기는 입출력조작을 DMA모듈에 넘기며 그러면 이 모듈이 그것을 담당 처리한다. DMA모듈은 한번에 한단어씩 전체 자료블로크를 이송시키는데 처리기를 거치지 않고 직접 기억기와 주고받기를 진행한다. 이송을 끝내면 DMA모듈은 처리기에 새치기신호를 보낸다. 이와 같이 이송의 시작과 끝점에서만 처리기를 참가시킨다(그림 1-19 ㄷ).

DMA모듈은 기억기에로 또 기억기로부터 자료를 이송시키기 위하여 모선을 조종해야 한다. 모선사용을 위한 경쟁때문에 처리기가 모선을 요구하고 DMA모듈을 기다려야 할 때가 있을수 있다. 이것은 새치기가 아니라는것 즉 처리기가 어떤 내용을 보관하지 않으며 그밖의 다른 일을 하지 않는다는것을 강조해 둔다. 다만 한 모선주기동안 처리기가 잠간 정지하는것뿐이다. 총적인 효과는 DMA이송기간에 처리기가 모선에 접속할것을 요구하는 경우에 처리기집행이 떠지게 된다는것이다. 그럼에도 불구하고 다중단어입출력이송에서 DMA는 새치기구동식이나 프로그람식입출력에 비하여 훨씬 더 효과적이다.

참 고 문 헌

[STAL00]은 이 장에서 론의된 문제들을 상세히 포괄하고 있다. 그밖에 콤퓨터방식과 조직에 대한 책들이 많다. 보다 가치 있는 책들중에는 다음과 같은것들이 있다. [PATT98]에서는 포괄적인 개괄을 주고 있으며 [HENN96]은 같은 저자가 집필한것으로서 설계의 많은 측면들을 강조해 주는 최근문헌이다.

HENN96 Hennessy, J., and Patterson, D. *Computer Architecture: A Quantitative Approach.* San Mateo, CA: Morgan Kaufmann, 1996.

PATT98 Patterson, D., and Hennessy, J. *Computer Organization and Design: The Hardware/Software Interface.* Sun Mateo, CA: Morgan Kaufmann, 1998

STAL00 Stallings, W. *Computer Organization and Architecture,* 5th ed. Upper Saddle River, NJ: Prentice Hall, 2000.

련 습 문 제

1. 그림 1-3의 가상기계는 두개의 입출력명령을 가지고 있다.

0011 = Load AC from I/O
0111 = Store AC to I/O

이 경우에 12bit 주소는 특정한 외부장치를 지정한다. 다음의 프로그람들의 집행과정(그림 1-4의 형태를 사용하여)을 보여 주시오.

① 장치 5으로부터 들어 오는 자료를 AC에 적재한다.
② 기억주소 940의 내용을 더한다.
③ AC의 내용을 장치 6에 기억시킨다.

장치 5에서 들어 오는 값은 3이며 기억주소 940의 내용은 값이 2라고 가정한다.

2. 그림 1-4에서 프로그람집행을 6개의 단계를 사용하여 문장으로 설명하고 있다. MAR와 MBR의 사용에 대하여 보여 줄수 있게 이 설명을 확장하시오.

3. 어떤 가상적인 32bit극소형처리기가 두개의 마당으로 구성된 32bit명령들을 가지고 있다고 하자. 즉 첫 바이트는 연산코드이며 나머지는 직접연산수 또는 연산수주소이다.

① 직접주소가능한 최대기억기용량은 얼마인가(바이트로 환산하여)?
② 극소형처리기모선들이

ㄱ) 32bit국부주소모선과 16bit국부자료모선 또는
ㄴ) 16bit국부주소모선과 16bit국부자료모선을 가지는 경우 체계속도에 미치는 영향을 설명하시오.

③ 프로그람계수기와 명령등록기에 몇개의 비트가 필요한가?

4. 16bit주소(실례로 프로그람계수기와 주소등록기들의 폭이 16bit라고 가정한다.)를 내보내며 16bit자료모선을 가지는 가상적인 극소형처리기를 고찰하자.

ㄱ)《16bit기억기》에 련결되는 경우 처리기가 직접접근할수 있는 최대기억주소공간은 얼마인가?

ㄴ)《8bit기억기》에 련결되는 경우 처리기가 직접접근할수 있는 최대기억주소공간은 얼마인가?

ㄷ) 극소형처리기가 개별적인 《입출력공간》에 접근하도록 하기 위한 구조적특징들은 무엇인가?

ㄹ) 입력과 출력명령이 8bit입출력포구번호를 지정할수 있다면 몇개의 8bit입출력포구가 극소형처리기를 지원할수 있는가? 16bit포구에 대하여서는 몇개인가를 설명하시오.

5. 16bit외부자료모선을 가지고 8MHz박자로 구동되는 32bit극소형처리기를 고찰하자. 극소형처리기가 최소 4개의 박자주기를 가지는 모선주기를 가지고 있다고 하자. 극소형처리기가 처리할수 있는 최대자료이송속도는 얼마인가? 극소형처리기의 성능을 증가시키기 위하여 외부자료모선을 32bit로 하는것이 더 좋겠는가 아니면 극소형처리기에 공급되는 외부박자를 2배로 하는것이 더 좋겠는가? 할수 있는 어떤 다른 가정을 설정하고 설명하시오.

6. 간단한 건반/인쇄전신기를 조종하는 입출력모듈을 포함하고 있는 콤퓨터체계를 고찰하자. 다음의 등록기들이 CPU에 있고 직접 체계모선에 련결되여 있다.

INPR: 입력등록기, 8bit
OUTR: 출력등록기, 8bit
FGI: 입력기발, 1bit
FGO: 출력기발, 1bit
IEN: 새치기가능, 1bit

인쇄전신기로부터의 건반입력과 인쇄기에로의 출력은 입출력모듈이 조종한다. 전신기는 자모수자식기호를 8bit단어로 부호화할수 있고 8bit단어를 자모수자식기호로 복호화할수 있다. 전신기로부터 입력등록기에 하나의 8bit단어가 들어갈 때 입력기발이 설정된다. 출력기발은 한단어가 인쇄될 때 설정된다.

ㄱ) 이 문제에서 제시된 첫 4개의 등록기를 사용하여 CPU가 인쇄전신기와의 입출력을 어떻게 할수 있는가를 설명하시오.

ㄴ) IEN을 사용하여 이 기능을 어떻게 더 효률적으로 수행할수 있는가를 설명하시오.

7. DMA모듈을 가지는 모든 체계들에서는 실제적으로 주기억기에 대한 DMA접근이 그에 대한 처리기접근보다 더 높은 우선권을 가진다. 그 리유는 무엇인가?

8. DMA모듈이 9600bps속도로 송신하고 있는 외부장치로부터 주기억기에로 문자들을 전송하고 있다. 처리기는 초당 백만개의 명령속도로 명령을 불러 낼수 있다. DMA의 동작으로 인하여 처리기의 속도가 얼마나 떠지게 되는가?

9. 콤퓨터가 CPU와 한 단어폭을 가진 자료모선과의 공유모선을 통하여 주기억기 M에 련결할수 있는 입출력장치 D로 구성되여 있다. CPU는 초당 최대 106개의 명령을 집행할수 있다. 명령은 평균 5개의 기계주기를 요구하며 그중 3개는 기억기모선을 사용한다. 기억기읽기 또는 쓰기조작은 하나의 기계주기를 사용한다. CPU가 95%의 명령집행속도를 요구하는《배경프로그람》을 련속적으로 집행하면서 입출력명령은 아무것도 집행하지 않는다고 하자. 한개의 처리기주기는 하나의

모선주기와 같다고 가정하자. 이제 상당히 큰 자료블로크가 M과 D사이에서 이송된다고 하자.

ㄱ) 프로그람식입출력을 사용하여 매개 한개 단어의 입출력이송이 CPU로 하여금 두개 명령을 집행할것을 요구하는 경우 D를 통하여 가능한 초당 단어수로서 최대입출력자료이송속도를 평가하시오.

ㄴ) DMA이송을 사용하는 경우 우와 같은 속도를 평가하시오.

10. 다음의 코드를 고찰하자.

```
for ( i = 0;  i < 20;  i ++ )
        for ( j = 0;  j < 10;  j ++ )
             a[ i ] = a[ i ] * j
```

ㄱ) 코드에서 공간적국소성에 대하여 하나의 실례를 드시오.

ㄴ) 코드에서 시간적국소성에 대하여 하나의 실례를 드시오.

11. 부록 1-ㄱ에서 방정식 1-1과 1-2를 n-준위기억기계층구조로 일반화하시오.

12. 다음의 파라메터를 가지는 기억기체계를 고찰하자.

T_c = 100ns　　C_c = 0.01쎈트/비트
T_m = 1,200ns　C_m = 0.001쎈트/비트

ㄱ) 1Mbyte주기억기의 비용은 얼마인가?

ㄴ) 캐쉬의 기능을 사용하는 1Mbyte주기억기의 비용은 얼마인가?

ㄷ) 캐쉬의 접근시간보다 유효접근시간이 10% 더 크다면 명중률 H는 얼마인가?

13. 콤퓨터가 캐쉬, 주기억기, 가상기억용디스크를 가지고 있다. 참조할 단어가 캐쉬안에 있으면 그것에 접근하는데 20ns가 요구된다. 그것이 캐쉬가 아니라 주기억기안에 있으면 그것을 캐쉬에로 적재하는데 60ns가 요구된다(이 시간은 초기에 캐쉬를 검사하는 시간이다.). 그때 참조를 시작한다. 단어가 주기억기에 없다면 디스크로부터 그 단어를 불러 내는데 12ms 걸리며 뒤따라 그것을 캐쉬에 복사하는데 60ns 걸리고 이어 참조를 다시 시작한다. 캐쉬명중률은 0.9이고 주기억기명중률은 0.6이다. 이 체계에서 참조할 단어에 접근하는데 평균접근시간이 ns로 얼마나 요구되는가?

14. 처리기가 수속호출과 복귀를 관리하기 위하여 탄창을 사용한다고 하자. 프로그람계수기로 탄창의 웃끝을 사용하면 프로그람계수기를 없앨수 있는가?

부록 1-ㄱ. 2준위기억기의 성능지표

이 장에서는 주기억기와 처리기사이에서 2준위내부기억기를 형성하면서 완충기로 작용하는 캐쉬에 대하여 고찰한다. 2준위방식은 국소성이라는 특성을 적용함으로써 1준위기억기에 비하여 성능을 훨씬 높여 준다. 이 부록에서는 이에 대하여 론의한다.

주기억기캐쉬기구는 콤퓨터방식의 한 부분으로서 장치적으로 실현되며 대체로 조작체계가 변경시킬수 없다. 따라서 이 책에서는 구체적으로 고찰하지 않는다. 그러나 조작체계에서 역시 국소성을 발전시켜 부분적으로 실현되는 2준위기억기방법에 대한 두가지 다른 실례 즉 가상기억기와 디스크캐쉬(표 1-2)가 있다. 이 두 문제는 제8장과 제11장에서 각각 론의한다. 이 부록에서는 세가지 방법에 공통적인 2준위기억기에 대한 몇가지

동작특성에 대하여 본다.

국소성

2준위기억기의 동작상 우점의 기본은 제5절에서 언급한 국소성의 원리이다. 이 원리는 기억기참조가 클라스터화되는 경향을 가진다는것을 설명하고 있다. 오랜 기간에는 사용중에 있는 클라스터가 변화되지만 짧은 기간에 처리기는 고정된 클라스터의 기억기를 참조하면서 동작한다.

국소성의 원리가 유효하다는것을 알수 있다. 다음의 몇가지 리유들을 고찰하자.

표 1-2. 2준위기억기의 특성

	주기억기캐쉬	가상기억기(페지화)	디스크캐쉬
대표적인 접근시간비	5:1	1000:1	1000:1
기억기관리체계	특수한 장치로 실현	장치와 체계프로그람결합	체계프로그람
대표적인 블로크크기	4~128Bytes	64~4096Bytes	64~4096Bytes
2차준위에 대한 처리기의 접근	직접접근	간접접근	간접접근

1. 모든 프로그람명령들중에서 단지 작은 부분을 이루는 갈래나 호출명령을 제외하고 프로그람집행은 순차적이다. 이로부터 대부분의 경우 즉시에 불러 내야 할 다음의 명령은 현재 불러 내는 명령뒤에 있다.
2. 후에 복귀하게 되는 호출이 오래동안 새치기되지 않는 경우는 드물다. 그러나 프로그람이 수속-호출깊이가 좁은 창문에 국한되는 경우가 더 흔하다. 그러므로 짧은 시간동안 명령들에 대한 참조는 몇개의 수속에 국부적으로 치우치는 경향성을 가진다.
3. 대부분의 반복되는 프로그람들은 여러번 반복되는 상대적으로 작은 몇개의 명령으로 구성된다. 따라서 반복이 지속되는동안 콤퓨터작업은 잇닿은 작은 프로그람부분에 국한되여 있다.
4. 많은 프로그람들에서 대부분의 계산은 배렬이나 레코드계렬과 같은 자료구조처리를 동반하고 있다. 많은 경우에 자료구조에 대한 련속적인 참조는 배치된 자료항목가까이에 있다.

이 리유들은 많은 연구들에서 확증되였다. 리유 1을 참조하여 많은 연구들에서 고급언어프로그람들의 성질을 분석하였다. 표 1-3은 다음의 연구들로부터 집행되는 동안 여러가지 상태의 출현을 측정한 주요결과들을 보여 주고 있다. 프로그람작성언어의 성질에 대한 최근 연구는 Knuth[KNUT71]이 진행하였으며 학생실습용으로 사용된 FORT RAN프로그람묶음들을 조사하였다. Tanenbaum [TANE78]은 조작체계프로그람들에 사용된 300개이상의 수속들로부터 수집한 측정자료들을 출판하였고 구조화프로그람(SAL)을 지원하는 언어를 작성하였다. Patterson과 Sequein[PATT82]는 번역프로그람들과 형설정프로그람콤퓨터지원설계(CAD)로부터 얻은 한조의 측정자료들을 분류하고 파일비교를 하면서 분석하였다. 또한 프로그람작성언어 C와 Pascal을 연구하였다. Huck [HUCK83]은 네가지 종류의 프로그람들을 분석하여 고속푸리에변환과 련립미분방정식의 적분을 포함한 혼합형일반목적과학계산프로그람을 내놓으려고 하였다. 여러가지 언어들과 응용결과에서 좋은 일치성을 찾아 볼수 있다. 즉 갈래 및 호출명령들이 프로그람의 수명기간에 몇분의 1밖에 나타나지 않았다는것이다. 이리하여 이 연구들은 선행목록에서 주장 1을 확증해 주고 있다.

표 1-3. 고급언어조작에 대한 상대적기능빈도

연구 **언어** **연구대상**	[HUCK83] Pascal 과학계산	[KMUT71] FORTRAN 실습	[PATT8] Pascal 체계	 C 체계	[TANE78] SAL 체계
Assign	74	67	45	38	42
Loop	4	3	5	3	4
Call	1	3	15	12	12
IF	20	11	29	43	36
GOTO	2	9	—	3	—
기타	—	7	6	1	6

주장 2에 대하여서는 [PATT85]에서 보고한 연구들이 확증해 준다. 이것을 그림 1-20에서 설명해 주고 있으며 이것은 호출-복귀동작을 보여 준다. 매개 호출은 오른쪽 아래로 이동하는 선으로, 매개 복귀는 오른쪽 우로 이동하는 선으로 표시되여 있다. 그림에서는 깊이가 5와 같은 창문을 정하고 있다. 둘중 어느 방향에서나 눈금이동이 6인 호출과 복귀계렬만이 창문을 이동시키고 있다. 알수 있는바와 같이 집행프로그람은 오랜 시간동안 정적인 창문내에 머물러 있을수 있다. C와 Pascal프로그람에 대해서 같은 분석을 진행한 연구결과는 호출과 복귀의 1%이하에 대해서만 깊이가 8인 창문을 이동시키게 된다는것을 보여 주었다[TAMI83].

참조의 국소성에 대한 원리는 최근 연구에서도 계속 확인되고 있다. 실례로 그림 1-21은 어떤 단일싸이트에 있는 Web페지호출패턴에 대한 연구결과들을 보여 주고 있다.

문헌에서는 공간적국소성과 시간적국소성을 명백히 구별하고 있다. **공간적국소성**은 클라스터화되는 몇개의 기억위치들에서 집행이 이루어 지는 경향성을 가리키는 말이다. 이것은 처리기가 명령들에 순차적으로 접근하게 된다는 경향성을 반영하고 있다. 공간적 기억위치는 또한 자료표를 처리할 때와 같이 프로그람이 자료기억위치에 순차적으로 접근하는 경향성을 가진다는것을 반영하고 있다. **시간적국소성**은 처리기가 최근에 사용한 기억기위치들을 호출하는 경향성을 가리키는 말이다. 실례로 반복고리를 집행할 때 처리기는 동일한 명령의 모임을 반복하여 집행한다.

시간
(호출/복귀를 단위로)
t= 33
복귀
호출
w = 5
겹치는 깊이

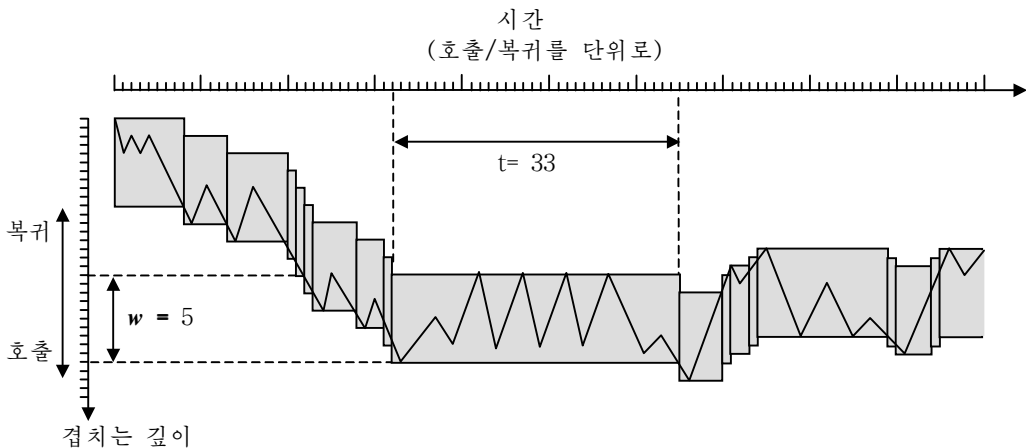



그림 1-20. 프로그람의 호출복귀거동의 실례

2준위기억기의 동작

국소성특성을 2준위기억기를 형성하는데 도입할수 있다. 웃준위기억기(M1)는 아래준위기억기(M2)보다 빠르며 값이 비싸다(비트당). M1은 큰 M2의 일부 내용을 림시 기억하는데 사용한다. 기억기를 참조할 때 M1에서 항목을 찾으려고 한다. 이것이 성공하면 고속접근이 이루어 진다. 그렇지 못하면 기억위치의 한 블로크를 M2에서 M1에로 복사하고 그다음 M1을 통하여 접근이 이루어 진다. 국소성의 원리로 하여 일단 블로크를 M1에 가져다 넣으면 블로크에 있는 기억위치들에 대한 접근이 여러번 진행되며 결국 총체적으로 고속봉사를 실현하게 된다.

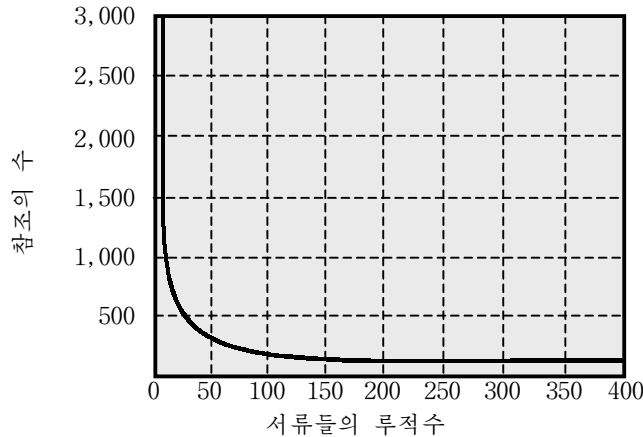



그림 1-21. Web페지들에서의 참조의 국소성[BAEN97]

평균항목접근시간을 표시하기 위하여 아래준위기억기의 속도뿐아니라 주어 진 참조가 M1에서 발견될수 있는 가능성도 고찰하여야 한다.

$$T_S = H \times T_1 + (1-H) \times (T_1 + T_2)$$
$$= T_1 + (1-H) \times T_2 \qquad (1-1)$$

여기서

T_S = 평균(체계)접근시간
T_1 = M_1의 접근시간(례로 캐쉬, 디스크캐쉬)
T_2 = M_2의 접근시간(례로 주기억기, 디스크)
H = 명중률(참조가 M1에서 발견되는 시간비률)

그림 1-15는 명중률함수로 표시한 평균접근시간을 보여 주고 있다. 알수 있는바와 같이 높은 명중률을 가지고 평균접근시간이 M2보다 M1에 훨씬 더 가깝다.

성능

2준위기억기기구를 평가하는데서 의의 있는 몇가지 파라메터를 고찰해 보자. 다음의 관계가 성립한다.

$$Cs = (C_1S_1 + C_2S_2) / (S_1 + S_2) \qquad (1-2)$$

여기서

Cs = 조합된 2준위기억기에서의 비트당 평균비용
C_1 = 웃준위기억기 M1의 비트당 평균비용

C_2 = 아래준위기억기 M2의 비트당 평균비용
S_1 = M1의 크기
S_2 = M2의 크기

$Cs \approx C_2$이면 좋다. $C_1 >> C_2$로 주어 지면 $S_1 << S_2$로 될것을 요구한다. 그림 1-22는 이 관계를 보여 주고 있다.[5]

다음은 접근시간을 고찰하자. 2준위기억기가 뚜렷한 성능개선을 가져 오자면 T_s가 대략적으로 T_1과 같아 져야 한다($Ts \approx T_1$). T_1이 T_2보다 훨씬 작게 주어 지면($T_1 \ll T_2$) 명중률은 1에 접근한다.

그러므로 우리는 M1을 작게, 비용을 낮추면서 명중률을 크게 개선하며 나아가서 성능을 개선하려고 한다. 합리적으로 확장하는데 필요한 두가지 요구를 만족시키는 M1의 크기가 있는가? 우리는 이 질문에 몇가지 보조질문을 가지고 대답할수 있다.

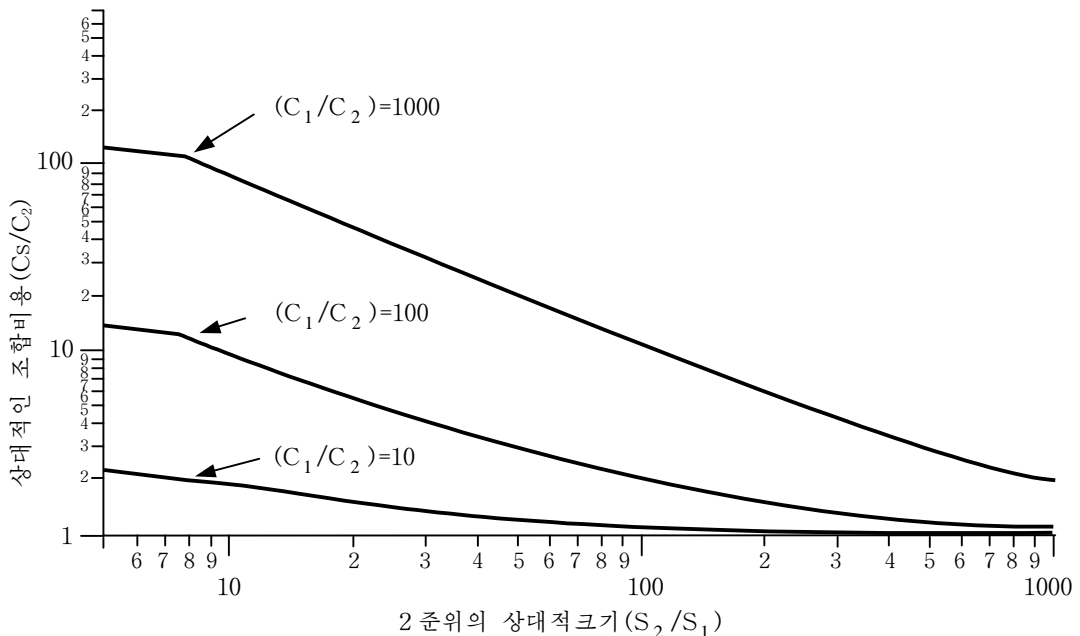



그림 1-22. 2준위기억기에서의 상대적기억기크기에 대한 평균기억기비용

- 동작상요구를 만족시키는데 필요한 명중률의 값은 얼마인가?
- 필요한 명중률을 보장하는 M1의 크기는 얼마인가?
- 그 크기가 비용에 대한 요구를 만족시키는가?

이것을 얻기 위해 접근효률이라고 하는 량 T_1/Ts를 고찰하자. 이것은 평균접근시간(Ts)이 M1접근시간(T_1)에 얼마나 가까운가를 보여 주는 량이다. 방정식 1-1로부터

$$T_1/T_2 = 1/[1+(1-H)T_2/T_1] \quad (1-3)$$

---

[5]· 축들은 모두 로그척도를 사용하고 있다. 로그척도에 대한 기초적설명은 Computer Sciencse Student Support Site at William Stallings.com/Student.Support.html 수학참고서에 있다.

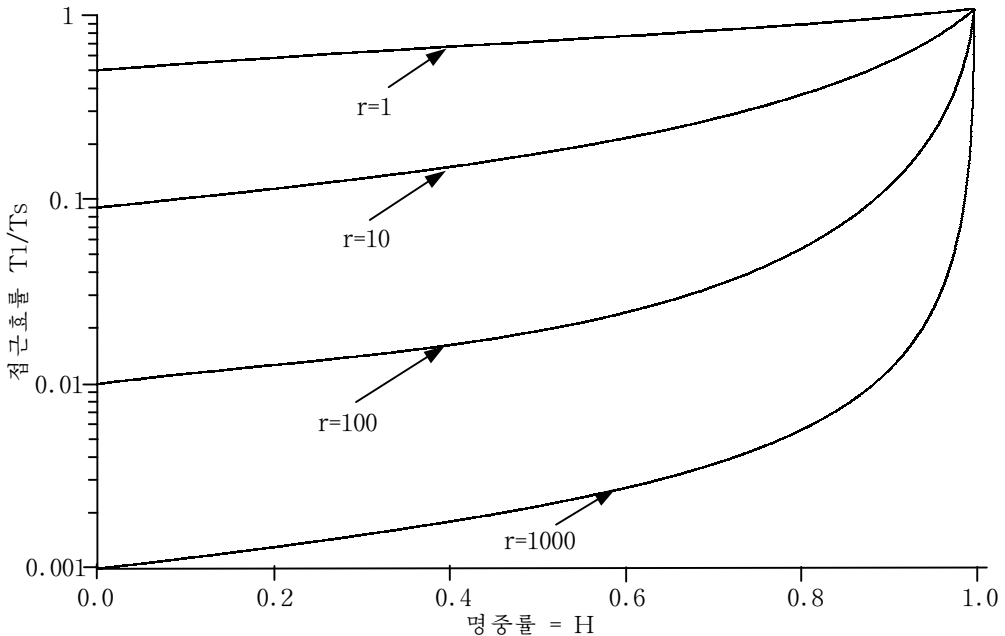



그림 1-23. 명중률의 함수로 표시한 접근효률($r=T_2$ / T_1)

그림 1-23에서는 T_1/Ts를 량 T_2 /T_1을 파라메터로 하고 명중률 H의 함수로 작도하였다. 대체로 캐쉬의 접근시간은 주기억기접근시간보다 5~10배 더 빠르며 주기억기접근시간은 디스크접근시간보다 대략 1000배 더 빠르다(T_1/T_2 = 1000). 이런데로부터 0.8~0.9범위내의 명중률이 성능요구를 만족시키는데 필요한것으로 볼수 있다.

이제는 상대적인 기억기크기에 대한 질문에 보다 정확히 대답할수 있다. 명중률이 0.8인가 또는 $S_1 \ll S_2$의 관계가 더 합리적인가? 이것은 집행되는 프로그람의 특징과 2준위기억기의 설계세부를 포함하는 몇가지 요인에 관계된다. 물론 주되는 판정기준은 국소

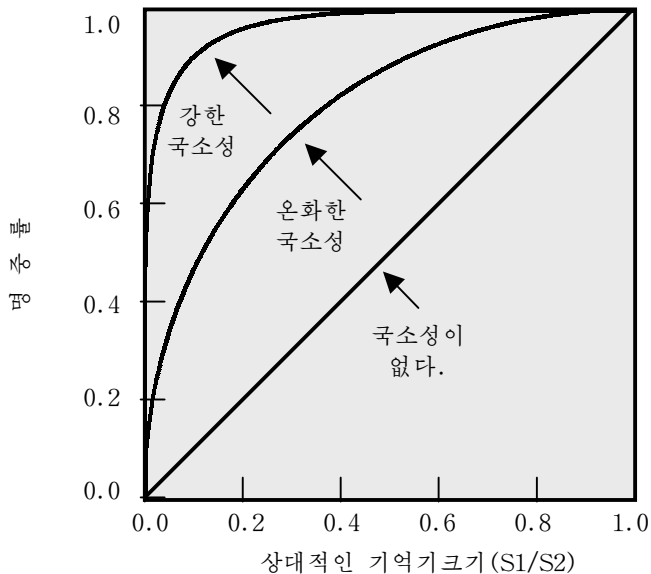



그림 1-24. 상대적인 기억기의 함수로 표시한 명중률

성의 정도이다. 그림 1-24는 명중률이 주어 진 조건에서 국소성의 효과를 보여 주고 있다. 명백한것은 M1이 M2와 같은 크기를 가지면 명중률은 1.0으로 된다는것 즉 M2의 모든 항목들이 항상 M1에도 기억되여 있다는것이다. 이제 아무런 국소성도 없다고 즉 참조가 완전히 우연적으로 진행된다고 가정하자. 실례로 M1이 M2의 절반크기를 가지면 임의의 시간에 M2의 절반항목이 M1에도 있게 되며 명중률은 0.5로 될것이다. 약한 국소성과 강한 국소성의 효과를 그림에서 보여 주고 있다.

강한 국소성이 있는 경우 상대적으로 작은 웃준위기억기크기로도 높은 명중률값을 얻을수 있다. 실례로 많은 연구들은 작은 캐쉬의 크기가 주기억기의 크기에 관계없이 0.75이상의 명중률을 보장하리라는것을 보여 주었다(실례로 [AGAR89], [PRZY88], [STRE83], [SMIT82]). 일반적으로 캐쉬는 1K~128K단어범위에 있으면 충분하며 반면에 주기억기는 현재 대체로 수메가바이트범위내에 있다. 가상기억기와 디스크캐쉬를 고찰할 때 같은 현상 즉 상대적으로 작은 M1이 국소성으로 하여 높은 명중률을 보장한다는것을 확증해 주는 다른 연구자료들을 인용하려고 한다.

이것은 목록에 있는 마감질문에로 우리를 이끌어 간다. 두 기억기의 상대적크기가 비용요구를 만족시키는가? 그 대답은 명백히《예》이다. 좋은 성능을 얻기 위하여 단지 상대적으로 작은 웃준위기억기를 요구한다면 2준위기억기의 비트당 평균비용은 값 눅은 아래준위기억기의 비용에 접근하게 된다.

부록 1-ㄴ. 수속조종

수속접근과 복귀의 집행을 조종하는 일반적인 기능은 탄창을 사용하는것이다. 이 부록은 탄창의 기초적인 속성을 요약하여 설명하고 수속조종에서 그의 사용에 대하여 고찰한다.

탄창실현

탄창은 요소들이 순차적으로 배렬된 모임으로서 한번에 그중 하나에만 접근할수 있다. 접근할 때에는 탄창의 꼭대기를 불러 내게 된다. 탄창내요소들의 수 즉 탄창의 길이는 가변적이다. 항목들은 단지 탄창의 꼭대기에 추가되거나 꼭대기에서부터 삭제될수 있다. 이로부터 탄창을 밀어넣기목록 또는 후입선출항목(LIFO)이라고도 한다.

탄창을 실현하는데서 어떤 기억위치모임을 탄창요소들을 기억시키는데 사용해야 한다. 대표적인 방법은 그림 1-25에서 설명하고 있다. 련속적인 기억위치를 가지는 블로크를 주기억기(가상기억기)에서 탄창용으로 예약한다. 대부분의 시간에 블로크는 탄창요소들을 부분적으로 채워 넣고 있으며 나머지부분은 탄창을 증가시키는데 사용할수 있다. 합리적으로 조작하자면 세개의 주소가 필요한데 이것들은 흔히 처리기등록기들에 기억되여 있다.

- **탄창지시기 :** 탄창의 꼭대기주소를 가리킨다. 항목을 탄창에 밀어 넣거나(PUSH) 밀어 낼 때(POP) 탄창지시기는 감소되거나 증가되여 탄창의 새로운 꼭대기주소를 가리킨다.
- **탄창기준 :** 예약된 블로크에서 바닥위치주소를 가리킨다. 이것은 항목을 빈 탄창에 밀어 넣을 때 사용되는 첫 기억위치이다. 탄창이 비여 있을 때 밀어 내면 오유를 통보한다.
- **탄창한계 :** 예약된 블로크의 다른 끝 즉 꼭대기주소를 가리킨다. 탄창이 꽉 차 있을 때 밀어 넣으려면 오유를 통보한다.

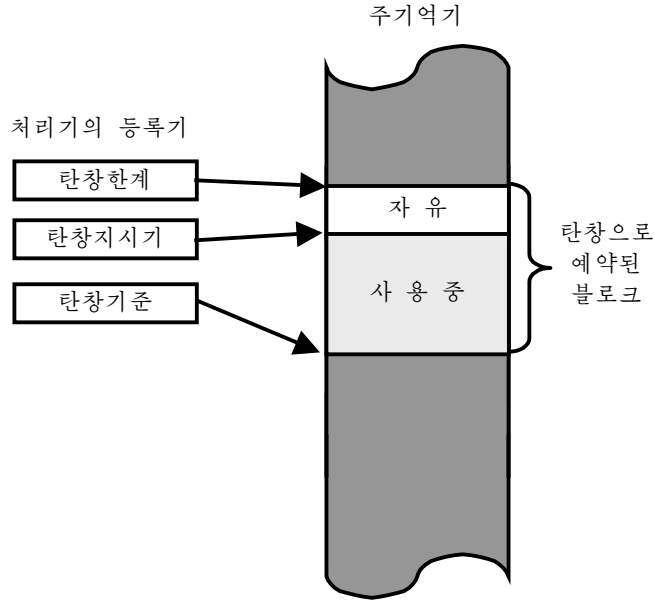



그림 1-25. 대표적인 탄창방식

전통적으로 그리고 오늘날 대부분의 기계들에서 탄창의 기준은 예약된 탄창블로크의 높은 주소끌에 있으며 한계는 낮은 주소끌에 있다. 이리하여 탄창은 보다 높은 주소로부터 보다 낮은 주소까지 장성한다.

수속호출 및 복귀

수속호출과 복귀를 관리하는 일반적수법은 탄창을 사용하는것이다. 처리기가 호출을 집행할 때 처리기는 복귀주소를 탄창에 넣는다. 복귀를 집행할 때 처리기는 탄창의 꼭대기에 있는 주소를 사용한다. 그림 1-26의 겹친수속에서의 탄창사용을 그림 1-27에서 설명하고 있다.

흔히 수속호출과 관련하여 파라메터들을 넘겨 주어야 한다. 이것을 등록기안에서 넘겨 줄수 있다. 다른 가능성은 파라메터들을 호출명령직후 기억기에 기억시켜 놓는것이다. 이 경우 복귀는 파라메터들 다음의 위치로 되여야 한다. 이 방법들은 둘다 약점이 있다. 등록기들을 사용한다면 호출되는 프로그람과 호출하는 프로그람을 등록기사용에서 이상이 없도록 작성하여야 한다. 기억기에 파라메터들을 기억시키는 방법은 파라메터의 수를 변화시키기 힘들게 한다.

파라메터를 넘겨 주는 보다 유연한 방법은 바로 탄창을 사용하는것이다. 처리기가 호출을 집행할 때 복귀주소를 탄창에 넣을뿐아니라 호출된 프로그람에 넘겨 주어야 하는 파라메터들도 탄창에 밀어 넣는다. 호출된 수속이 탄창에서 파라메터들을 호출할수 있다. 복귀할 때에도 반환파라메터들을 복귀주소밑에 들어 가도록 탄창에 밀어 넣을수 있다. 수속을 위하여 기억되는 복귀주소를 포함한 전체 파라메터묶음을 **탄창프레임**이라고 한다.

그림 1-28에 한가지 실례를 보여 주고 있다. 이 실례는 국부변수 X1과 X2를 선언하는 수속 P와 P로부터 호출을 받을수 있으며 그안에서 국부변수 Y1과 Y2를 선언하는 수속 Q를 보여 준다. 이 그림에서 매개 수속의 복귀점은 대응하는 탄창들에 기억되여 있는 첫 항목이다. 다음 앞틀의 시작과 관련한 지시기를 기억시킨다. 이것은 탄창에 들어가는 파라메터들의 수 즉 길이가 변할수 있기때문에 필요하다.

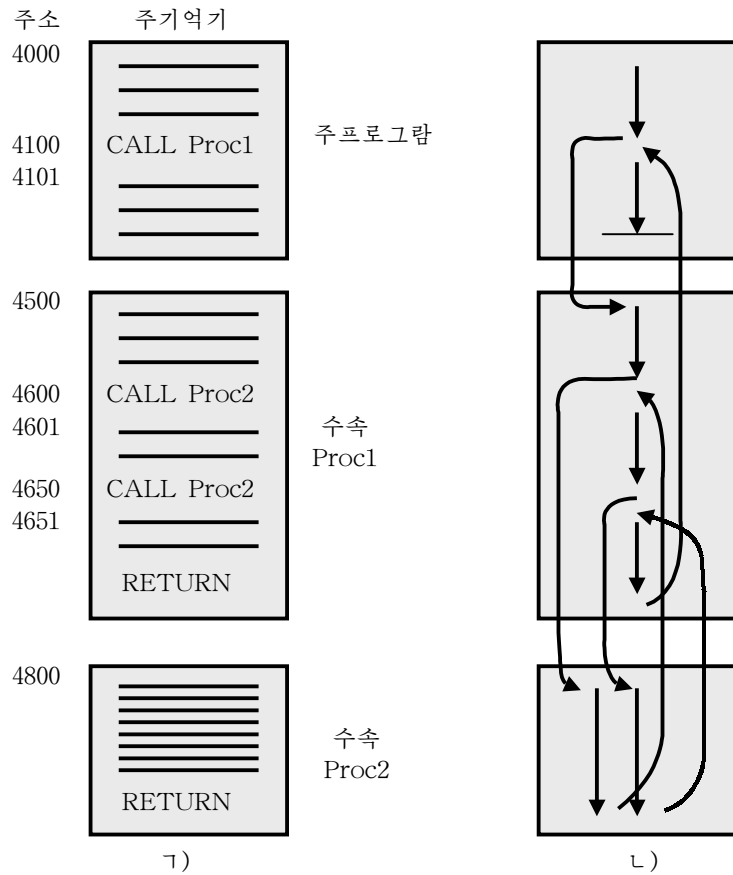



그림 1-26. 겹침수속들: ㄱ-호출과 복귀, ㄴ-집행순차

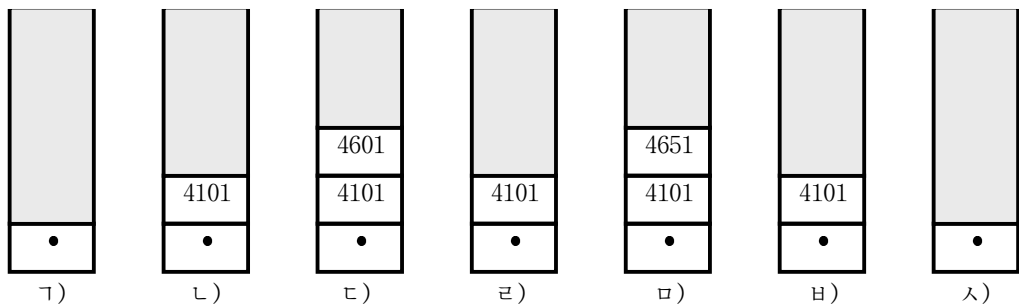



그림 1-27. 그림 1-26의 겹침수속을 실현하는데서 탄창의 사용
ㄱ-초기탄창내용, ㄴ-CALL Proc2후, ㄷ-초기 CALL Proc2, ㄹ-RETURN후,
ㅁ-CALL Proc2후, ㅂ-RETURN후, ㅅ-RETURN후

재진입가능한 수속

다중사용자를 지원하는데서 특히 유용한 개념은 재진입가능한 수속이다. 재진입가능한 수속은 같은 시간주기동안에 다중사용자가 단일한 프로그람복사를 공유할수 있다. 재진입가능성은 두가지 주요문제를 가지고 있다. 즉 프로그람코드 그자체는 변경시킬수 없으며 매개 사용자용국부자료는 따로따로 기억시켜야 한다는것이다. 재진입가능한 수속은 새치기프로그람이 새치기할수 있으며 호출할수 있다. 그리고 집행은 정확히 수속에로 복귀된다. 공유체계에서 재진입가능성은 주기억기를 보다 효과적으로 사용할수 있게 한다. 즉 한 복사의 프로그람코드가 주기억기에 있지만 하나이상의 응용프로그람들이 수속을 호출할수 있다.

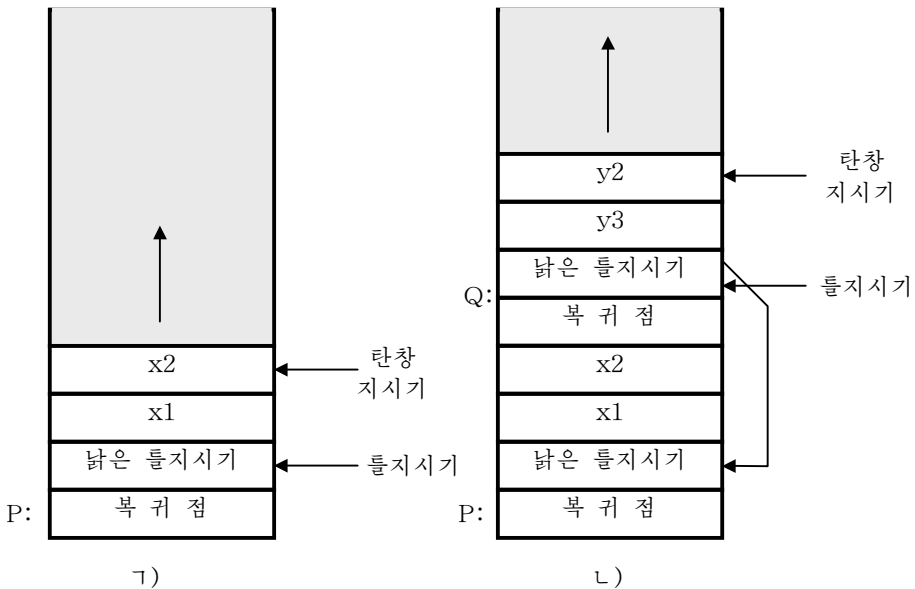



그림 1-28. 대표적인 수속 P와 Q를 사용하는데서 탄창프레임성장
ㄱ-P는 능동, ㄴ-P가 Q를 호출

그러므로 재진입가능한 수속은 영구부분(수속을 형성하는 명령들)과 림시부분(프로그람이 사용한 국부변수들을 호출하는 프로그람은 물론 기억기에 돌려 주는 지시기)을 가져야 한다. 활성화라고 하는 수속의 매개 집행요구는 영구부분에 있는 코드를 집행하게 되지만 자체의 국부변수와 파라메터들을 가져야 한다. 특정한 활성화와 관련되는 림시부분을 활성화레코드라고 한다.

재진입가능한 수속을 지원하는 가장 편리한 방법은 탄창을 사용하는것이다. 수속을 호출할 때 수속의 활성화레코드를 탄창에 기억시킬수 있다. 따라서 활성화레코드는 수속호출상에서 창조되는 탄창프레임의 한 부분으로 된다.

제 2 장. 조작체계의 개괄

이 장에서는 조작체계들의 주요개발과정을 설명한다. 개발과정은 이 분야의 전망을 보여 주며 또한 조작체계의 원리들을 개괄하는 목적에도 부합된다. 먼저 조작체계의 목적과 기능들을 고찰한다. 이어 원시적인 일괄체계로부터 복잡한 다중방식, 다중사용자체계에 이르기까지 조작체계가 어떻게 발전하여 왔는가를 고찰한다. 끝으로 이 책전반을 관통하고 있는 실례들로서 두개 조작체계의 개발과정과 일반적인 특징을 고찰한다.

제 1 절. 조작체계의 목적과 기능

조작체계는 응용프로그람의 집행을 조종하며 응용프로그람과 콤퓨터장치사이의 대면부로 동작하는 한개의 프로그람이다. 그의 목적은 세가지이다.

- **편리성 :** 콤퓨터사용을 보다 편리하게 하는것
- **효률성 :** 콤퓨터체계자원들을 효률적인 방식으로 사용할수 있게 하는것
- **확장능력 :** 봉사와 관련이 없이 새로운 체계기능들을 효과적으로 개발,시험, 도입할수 있게 구성하는것

이제부터 조작체계를 이 세가지 면에서 고찰하여 보자.

사용자/콤퓨터대면부로서의 조작체계

사용자에게 응용프로그람을 제공하는데 쓰이는 장치와 프로그람을 그림 2-1에서와 같이 층을 이루거나 계층구조로 된 형식으로 볼수 있다. 일반사용자인 응용프로그람의 사용자는 일반적으로 세부적인 콤퓨터장치와 관계가 없다. 일반사용자는 콤퓨터체계를 응용프로그람의 모임으로 본다. 응용프로그람은 프로그람작성언어로 표현할수 있으며 응용프로그람작성자가 개발한다. 만일 사람들이 콤퓨터장치를 원만히 조종할수 있는 기계명령들의 모임으로 응용프로그람을 개발한다고 하면 상당히 복잡한 과제에 부닥치게 될것이다. 이 과제를 쉽게 하기 위하여 체계프로그람들이 나왔다. 이런 프로그람들을 편의프로그람이라고 한다. 이 프로그람들은 흔히 프로그람참조, 파일관리, 입출력장치조종을 방조하는 기능들을 사용하였다. 프로그람작성자는 이 기능들을 사용하여 응용프로그람을 개발하며 응용프로그람은 실행기간에 편의프로그람들을 사용하여 일정한 기능들을 수행하게 된다. 가장 중요한 체계프로그람은 조작체계이다. 조작체계는 프로그람작성자로 하여금 장치의 세부를 모르고 체계를 사용하는데 편리한 대면부를 보장해 준다. 그것은 중계자로 동작하면서 프로그람작성자와 응용프로그람이 설비와 봉사기능을 더 쉽게 호출하여 사용할수 있도록 해준다.

조작체계는 주로 다음과 같은 령역들에서 봉사한다.

- **프로그람개발 :** 조작체계는 편집기와 오유수정기와 같은 설비와 봉사를 주어 프로그람창조에서 프로그람작성자를 방조한다. 대체로 이 봉사들은 편의프로그람형태로 되여 있으며 그것이 엄밀하게 조작체계의 핵심부분은 아니지만 조작체계와 함께 제공되므로 응용프로그람개발도구라고 한다.
- **프로그람집행 :** 프로그람을 집행하기 위하여 많은 과제들이 수행되여야 한다. 명령들과 자료가 주기억기안에 적재되여야 하며 입출력장치들과 파일들이 초기화되여야 하고 다른 자원들을 준비해야 한다. 조작체계는 사용자를 위하여 이 일정작성임무들을 조종한다.

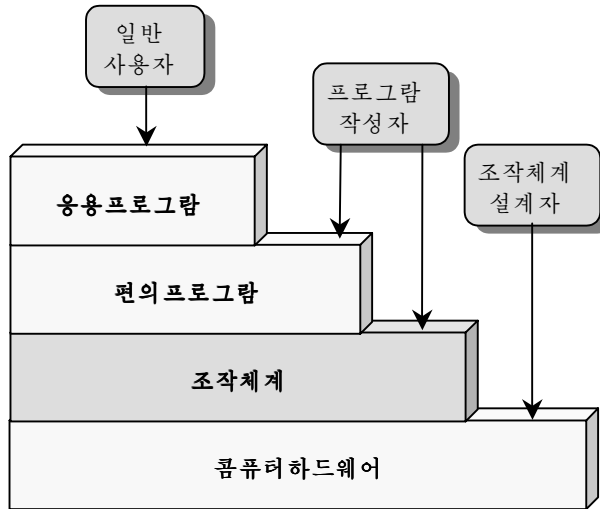



그림 2-1. 콤퓨터체계의 층들과 전경

- **입출력장치접근 :** 매개 입출력장치들은 조작을 위한 자기자체의 특유한 명령모임이나 조종신호를 요구한다. 조작체계는 이것들을 교갑화하고 있는 대면부를 제공하여 프로그람작성자가 단순한 읽기나 쓰기로 장치들에 접근할수 있도록 해준다.
- **파일접근조종 :** 파일들의 경우에 조작체계에서의 조종은 입출력장치의 특징(디스크구동, 테프구동)뿐아니라 기억매체우에 있는 파일들에 포함된 자료의 구조도 구체적으로 포함하고 있어야 한다. 더 나아가서 다중사용자를 가지는 체계인 경우에 조작체계는 파일들에 대한 접근조종을 하기 위하여 보호기구를 제공해야 한다.
- **체계접근 :** 공유체계나 공동체계에서 조작체계는 총적으로 체계에 대한 접근과 특정한 체계자원에 대한 접근을 조종한다. 접근기능은 허용되지 않은 사용자들로부터 자원과 자료를 보호해야 하며 자원경쟁에서 충돌문제를 해결하여야 한다.
- **오유검출과 응답 :** 콤퓨터체계가 동작할 때 여러가지 오유가 발생할수 있다. 이것은 기억기오유나 장치고장과 같은 내부장치적오유들과 연산자리넘침, 금지된 기억위치에 대한 접근시도, 응용프로그람의 요청에 조작체계가 보증할수 있는 능력의 부족과 같은 여러가지 프로그람적오유들을 포함한다. 매개 경우에 조작체계는 응답하여 응용프로그람집행상에서 최소한의 영향을 가지는 오유조건을 명백히 해주어야 한다. 응답은 오유를 발생시킨 프로그람을 새치기시키는데로부터 조작을 다시 하게 하거나 응용프로그람에 오유를 단순히 보고하는것으로 분류할수 있다.
- **회계 :** 좋은 조작체계는 여러가지 자원이나 응답시간과 같은 감시기성능파라메터들의 사용상태를 통계적으로 수집한다. 임의의 체계에서 이 정보는 앞으로의 증강요구를 예상하는데서와 체계성능을 개선하는데 사용할수 있다. 다중사용자체계에서 이 정보는 료금계산목적으로 사용할수 있다.

자원관리자로서의 조작체계

콤퓨터는 자료의 이동, 기억, 처리 그리고 이 기능을 조종하기 위한 자원들의 모임이다. 조작체계는 이 자원들을 관리할 책임을 지고 있다.

자료의 이동, 기억, 처리를 조종하는것이 바로 조작체계라고 말할수 있는가? 한쪽 측

면에서는 옳다. 즉 콤퓨터의 자원을 관리함으로써 조작체계는 콤퓨터의 기본기능을 조종하는 상태에 놓이게 된다. 그러나 이 조종은 좀 특이한 방법으로 진행된다. 보통 사람들은 조종기구라고 하면 어느것을 조종하는것의 외적인것, 즉 적어도 어느것을 조종대상과 명백하게 구별되는 부분이라고 생각한다(실례로 내부가열체계는 열원이나 열분배기구와는 완전히 구별되는 열전대가 조종한다.). 조작체계의 경우에는 사정이 다르다. 조작체계는 두가지 견지에서 일반적인 조종기구와 차이난다. 즉

- 조작체계기능들은 보통 콤퓨터쏘프트웨어와 같은 방식으로 되여 있으면서 처리기가 집행하는 프로그람이나 프로그람모임이라는것이다.
- 조작체계는 흔히 조종을 넘기거나 회복하는것을 처리기에 의존해야 한다는것이다.

조작체계는 사실상 한개의 콤퓨터프로그람모임외에 아무것도 아니다. 다른 콤퓨터프로그람들과 류사하게 그것은 처리기에 명령들을 준다. 주요한 차이는 프로그람의 목적에 있다. 조작체계는 처리기에 다른 체계자원을 사용하며 다른 프로그람의 집행에 들어 갈것을 지시한다. 그러나 처리기가 이러한것들을 하자면 조작하고 있는 체계프로그람집행을 중지하고 다른 프로그람들을 집행하여야 한다. 이와 같이 조작체계는 처리기가 어떤 《유용한》 일을 하도록 조종을 넘겨 주며 처리기가 다음일을 하기 위한 준비를 갖추는 시간동안 조종을 회복한다. 이 모든것에 포함되는 기구들은 앞의 장들에서 한 설명으로부터 명백하다.

그림 2-2는 조작체계가 관리하는 주요자원들을 보여 주고 있다. 조작체계의 일부분은 주기억기안에 있다. 이것이 **핵심부** 즉 **알맹이**이다. 이것은 주어 진 시간에 조작체계에서 가장 흔히 쓰이는 부분들과 현재 사용중에 있는 조작체계의 다른 부분들이다. 주기억기의 나머지부분에는 다른 사용자프로그람과 자료가 들어 있다. 자원(주기억기)의 배정은 조작체계와 처리기의 기억기관리장치에 의하여 공동으로 조종된다. 조작체계는 집행상태에 있는 프로그람이 언제 입출력장치를 사용할수 있는가를 결정하며 파일에 대한 접근과 그 사용을 조종한다. 처리기자체도 한개의 자원이며 조작체계는 특정한 사용자프로그람을 집행하는데 처리기시간이 얼마나 들어가는가 하는것을 판정하여야 한다. 다중처리기체계인 경우 처리기모두에 대하여 이 판정을 하여야 한다.

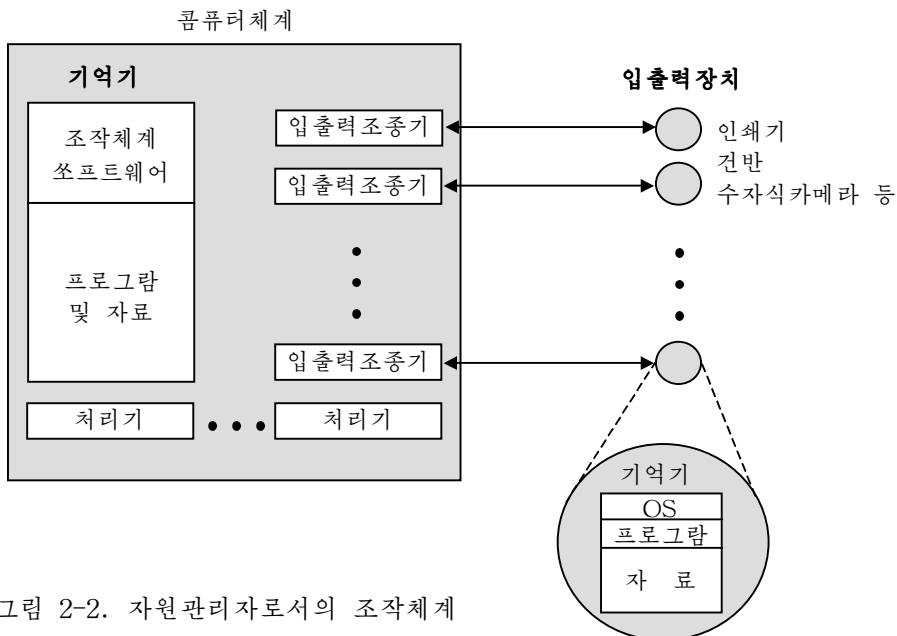



그림 2-2. 자원관리자로서의 조작체계

조작체계발전의 용이성

중요한 조작체계는 몇가지 리유로 하여 시간이 감에 따라 발전하게 된다. 즉

- **장치갱신과 새로운 형의 하드웨어 :** 실례로 UNIX와 IBM OS/2조작체계의 초기 판본들은 페지호출프로그람을 사용하지 못하였다. 그것은 이 판본들이 페지화하드웨어가 없는 기계들에서 집행되였기때문이다.[1] 최근 판본들은 페지화능력을 발휘하도록 수정되였다. 또한 1회1행흘리기방식대신에 도형말단이나 페지방식을 사용하면 조작체계설계에 영향을 줄수 있다. 실례로 말단은 사용자가 화면상의 《창문》들을 통하여 같은 시간에 몇가지 응용프로그람들을 관찰하도록 할수 있다. 이것은 조작체계가 보다 복잡한 지원을 해줄것을 요구한다.
- **새로운 봉사들 :** 사용자요구에 응답하여 또는 체계관리기의 요구에 응답하여 조작체계는 새로운 봉사들을 줄수 있도록 확장해야 한다. 실례로 현존도구들을 가지고 사용자에게 좋은 성능을 계속 보장하기 힘들다는것이 알려 지면 조작체계에 조종도구들을 추가하여야 한다. 다른 실례는 현시화면상에서 창문들을 사용하려는 새로운 응용프로그람들이다. 이것은 조작체계에서 중요한 갱신을 요구한다.
- **곤경 :** 어떤 조작체계든지 결함들을 가지고 있다. 이것은 오랜 시간에 걸쳐서 발견되며 곤경에 빠지게 된다. 물론 그 곤경은 새로운 결함들을 초래할수 있다.

조작체계를 정기적으로 변화시킬데 대한 요구는 설계상의 일정한 요구로 된다. 명백히 말할수 있는것은 체계가 구조상 모듈들사이에서 명확히 정의된 대면부를 가진 모듈이라는것이며 문서화가 잘 되여 있어야 한다는것이다. 대표적인 현대 조작체계와 같은 큰 프로그람들에서도 모듈화가 옳바로 되였다고 보기에는 불충분하다[denn80 a]. 프로그람들을 단순히 보조루틴으로 분할하는것보다 훨씬 더 많은것을 해야 한다. 이 문제는 이 장의 뒤에서 보기로 한다.

제 2 절. 조작체계의 발전과정

조작체계에서의 주되는 요구들과 현대적인 조작체계의 중요특징들이 가지는 의의를 리해하자면 조작체계가 여러해동안 어떻게 발전하여 왔는가를 고찰해 보는것이 좋다.

직렬처리

1940년대 후반기부터 1950년대 중엽까지 개발된 최초의 콤퓨터들에서는 프로그람작성자들이 콤퓨터하드웨어와 직접 대화를 진행하였다. 즉 아무런 조작체계도 없었다. 이 기계들은 현시표시등들, 빗장스위치들, 일부 입력장치와 인쇄기로 구성된 조종탁으로 동작하였다. 기계코드로 된 프로그람들이 입력장치(실례로 카드읽기장치)를 거쳐 입력되였다. 오유가 있으면 프로그람을 정지시켰으며 오유조건은 표시등들에 표시되였다. 프로그람작성자는 오유의 원인을 판단하기 위하여 처리기등록기들과 주기억기를 조사할수 있었다. 프로그람이 정상완료되면 인쇄기상에 출력이 인쇄되였다.

이 초기체계들은 두개의 문제점들을 제기하였다.

- **일정작성 :** 대부분의 설비들에서는 기계시간을 예약하기 위하여 경복사기호판을 사용하였다. 대체로 사용자는 반시간정도의 배수간격으로 한개의 시간블로크동안 기호화를 할수 있었다. 어떤 사용자는 한시간동안 기호화를 하고 45min동안에 끝낼수도 있었다. 이것은 콤퓨터를 많은 시간 휴식상태에 놓이게 하였다. 다른

---

[1] 페지화는 이 장의 뒤에서 설명하며 제 7 장에서 구체적으로 설명한다.

한편 사용자는 문제들을 파고 들어야 했고 배정된 시간에 끝내지 못할수도 있었으며 문제를 해결하지 못한채 끝마칠 때도 있었다.

- **설치시간 : 일감**이라고 하는 단일한 프로그람은 콤파일러와 함께 고급언어프로그람(원천프로그람)을 기억기에 적재하고 번역된 프로그람(목적프로그람)을 보관하며 목적프로그람과 공통기능들을 함께 넣고 련결하는 과정을 포함하였다. 이 매 단계들은 테프의 설치와 해제, 카드조설치를 동반하였다. 오유가 발생하면 사용자는 되돌아 설치순차를 다시 밟아야 하였다. 따라서 프로그람을 설치하고 실행시키는데 많은 시간을 소비하였다.

이 조작방식은 사용자들이 콤퓨터에 직렬로 접근한다는것을 반영하여 직렬처리라고 말할수 있었다. 시간이 흐르면서 여러가지 체계프로그람들이 개발되여 직렬처리를 더 효과적인 처리로 만들려고 하였다. 이 도구들은 모든 사용자들이 공동으로 사용할수 있는 공통기능부, 련결편집기, 적재프로그람, 오유수정기, 입출력구동루틴들의 서고들이다.

단순일괄체계

초기 기계들은 값이 매우 비쌌고 따라서 기계의 유용성을 최대화하는것이 중요하였다. 일정작성과 설치시간으로 인하여 랑비되는 시간을 허용할수 없었다.

유용성을 개선하기 위하여 일괄조작체계에 대한 개념이 제기되였다. 첫 일괄조작체계(첫 조작체계라고 할수 있다.)를 IBM 701에서 사용하기 위하여 General Motors회사가 1950년대 중엽에 개발하였다[WEIZ81]. 이 개념은 후에 더욱 세련되였고 IBM구매자들에 의하여 IBM 704에서 실현되였다. 1960년대초에 판매자들이 자기들의 콤퓨터체계용으로 일괄조작체계를 개발하였다. 7090/7094콤퓨터용 IBM조작체계인 IBSYS는 다른 체계들에 그 영향이 널리 퍼진것으로 하여 특별히 지적할만하다.

단순일괄체계방안리면에 있는 중심사상은 **감시기**라고 하는 한개의 프로그람을 사용한다는데 있다. 이 형태의 조작체계를 사용함으로써 사용자는 더이상 기계에 직접 접근하지 않게 되였다. 이제는 사용자가 카드나 테프상에 일감을 적어서 콤퓨터운영자에게 제출하면 운영자가 그 일감들을 순차적으로 한데 묶어 감시기가 사용하도록 입력장치에 총적으로 묶어 넣으면 되였다. 매개 프로그람은 처리를 끝내면 감시기에 되돌아 가도록 구성되여 있다. 그 시점에서 감시기는 자동적으로 다음프로그람을 적재하기 시작한다.

이 방안이 어떻게 동작하는가를 리해하기 위하여 그것을 두가지 견지 즉 감시기의 견지에서와 처리기의 견지에서 고찰하여 보자. 즉

s

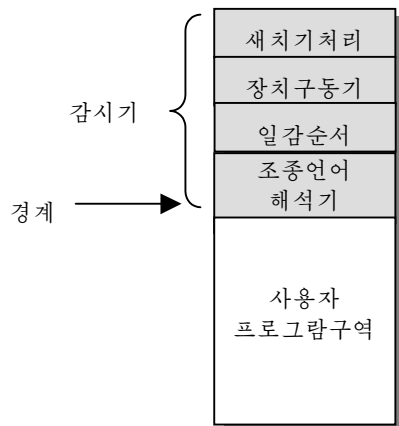



그림 2-3. 상주감시기의 기억기배치

- **감시기의 견지 :** 사건들의 순서를 조종하는것이 바로 감시기이다. 이렇게 되자면 관리기는 거의나 주기억기안에 항상 있어야 하며 집행할수 있어야 한다(그림 2-3). 이 부분을 **상주감시기**라고 한다. 감시기의 나머지부분은 일감을 시작할 때 사용자프로그람에서 보조루틴으로서 적재되는 편의프로그람과 공통기능부분들로 구성되여 있다. 감시기는 입력장치(대체로 카드읽기장치나 자기테프장치)로부터 일감을 한번에 한개씩 읽어 들인다. 읽어 들일 때 현재일감은 사용자프로그람구역에 배치되며 조종은 이 일감에로 넘어 간다. 일감들을 끌내면 조종은 감시기에 넘어 가며 그것은 즉시에 다음일감을 읽어 들인다. 매개 일감의 결과를 사용자에게 전달하기 위하여 인쇄기와 같은 출력장치에 내보낸다.
- **처리기의 견지 :** 일정한 시점에서 처리기는 감시기가 보관된 주기억기의 일부분에 있는 명령들을 집행한다. 일단 일감을 읽어 들이면 체계는 감시기안에 있는 갈래명령과 맞다들리는데 갈래명령은 사용자프로그람의 시작부에서부터 명령을 집행해 나가도록 처리기에 명령한다. 처리기는 완료되거나 또는 오유조건과 맞다들릴 때까지 사용자프로그람에서 명령을 집행한다. 어느 경우에나 처리기가 다음명령을 감시기프로그람에서 불러 내도록 한다. 따라서《조종이 일감에로 넘어 간다.》는 말은 간단히 처리기가 사용자프로그람에서 명령을 불러 내고 집행한다는것을 의미하며 《조종이 감시기에로 복귀된다》는 말은 처리기가 감시기프로그람에서 명령을 불러 내고 집행한다는것을 의미한다.

감시기가 일정작성문제를 조종하는것을 보자. 일괄일감들은 대기렬을 이루며 아무런 휴식시간도 없이 가능한 빨리 집행된다. 감시기는 물론 일감설정시간을 개선시킨다. 매개 일감에서 명령들은 **일감조종언어**(JCL)의 원시적인 형태를 가진다. 이것은 감시기에 명령들을 주기 위하여 사용되는 특수한 형태의 프로그람작성언어이다. 간단한 실례로서 프로그람작성언어 FORTRAN으로 작성한 사용자의뢰프로그람과 프로그람이 사용할 자료를 들수 있다. FORTRAN명령과 자료는 모두 개별적인 착공카드나 테프의 개별적인 레코드상에 있다. FORTRAN과 자료선들외에 일감은 일감조종명령들을 포함하는데 이것들은 시작을 《$》로 표기한다. 일감의 총적인 형태는 다음과 같이 볼수 있다. 즉

일감을 집행하기 위하여 감시기는 $FTN행을 읽고 자기의 대용량기억기(보통 테프)에서 해당한 언어콤파일러를 적재시킨다. 콤파일러는 사용자프로그람을 목적코드로 번역하며 그것을 주기억기나 대용량기억기에 기억시킨다. 주기억기에 기억된다면 그 조작을 《번역, 적재, 집행》이라고 부른다. 만일 테프에 기억된다면 $LOAD명령이 요구된다. 이 명령을 감시기가 읽고 번역조작한후에 조종을 다시 회복한다. 감시기는 적재프로그람을 불러 내여 목적프로그람을 기억기에 넣고(콤파일러가 있던 위치에) 조종을 그것에로 이행시킨다. 이런 방법으로 한번에 한개의 부분체계만을 집행시킬수 있다해도 서로 다른 부분체계들사이에서 주기억기의 큰 토막을 공유시킬수 있다.

사용자프로그람의 집행기간에 입력명령은 한행의 자료를 읽고 사용자프로그람에 있는 입출력명령은 사용되는 조작체계의 일부분으로 되여 있는 입력루틴을 기동시킨다. 입력루틴은 프로그람이 JCL행을 제대로 읽어 들이는가를 검사한다. 사고가 생기면 오유가 발생하며 조종은 감시기에로 넘어 간다. 사용자일감이 성공적으로 완료하든 못하든 감시기는 다음 JCL명령을 만날 때까지 입력행을 주사한다. 이렇게 함으로써 너무 많거나 너무 적은 자료행들을 가지는 프로그람을 보호하게 된다.

감시기 또는 일괄조작체계는 간단히 말하면 한개의 콤퓨터프로그람이다. 조종을 번갈아 처리하고 넘기는것은 처리기가 주기억기의 각이한 부분들에서 명령들을 불러 낼수 있는 능력에 관계된다. 여기에서 일부 다른 하드웨어적인 기능들이 요구된다. 즉

- **기억기보호 :** 사용자프로그람이 집행중에 있을 때 감시기가 있는 기억기구역을 변경시키지 말아야 한다. 그러한 시도가 있다면 처리기하드웨어는 오유를 검출하고 조종을 감시기에 이행한다. 그러므로 감시기는 일감처리에서 실패하고 오유통보를 내보내며 다른 일감을 적재한다.
- **시계 :** 시계는 한개의 일감이 체계를 독점하는것을 막는데 쓰인다. 시계는 매개 일감을 시작할 때 설정된다. 시계가 만기되면 사용자프로그람은 중지되고 조종은 감시기에로 돌아 간다.
- **특권명령 :** 일정한 기계준위명령들은 지정된 특권을 가지고 있으며 감시기만이 집행할수 있다. 사용자프로그람을 집행하는 동안 처리기가 이러한 명령과 만나면 오유가 발생하여 조종을 감시기에 이행시킨다. 특권을 가진 명령들속에 입출력명령들이 속하며 그런 경우 감시기는 입출력장치들에 대한 조종을 보류한다. 실례를 들면 사용자프로그람이 다른 일감으로부터 일감조종명령을 우연적으로 읽어 들이는것을 막는다. 사용자프로그람이 입출력을 수행하려고 한다면 그것은 감시기가 그것에 대한 조작을 진행하도록 요청해야 한다.
- **새치기 :** 사용자프로그람에 조종을 넘기거나 사용자프로그람으로부터 조종을 회복하는데서 더 많은 유연성을 가지게 한다.

물론 이러한 기능이 없는 조작체계를 만들수 있다. 그러나 콤퓨터판매자들은 결과를 예측할수 없다는것을 재빨리 알아 차리고 비교적 원시적인 일괄조작체계에도 이러한 장치적기능을 실현하였다.

일괄조작체계에서 기계시간은 사용자프로그람의 집행과 감시기집행사이에서 번갈아 소비된다. 이때 두가지 손실경향이 나타났는데 그것은 일부 주기억기가 감시기를 필요이상 차지하고 있다는것과 감시기에 의하여 일부 기계시간이 소비된다는것이였다. 이것은 둘다 간접소비시간을 발생시킨다. 이 간접소비시간이 있음에도 불구하고 단순일괄체계는 콤퓨터의 사용을 개선한다.

파일로부터 레코드읽기	0.0015s
100개의 명령집행	0.0001s
파일로부터 레코드쓰기	0.0015s
계	0.0031s

CPU사용퍼센트 ＝ 0.0001/ 0.0031＝0.032＝3.2％

그림 2-4. 체계사용률실례

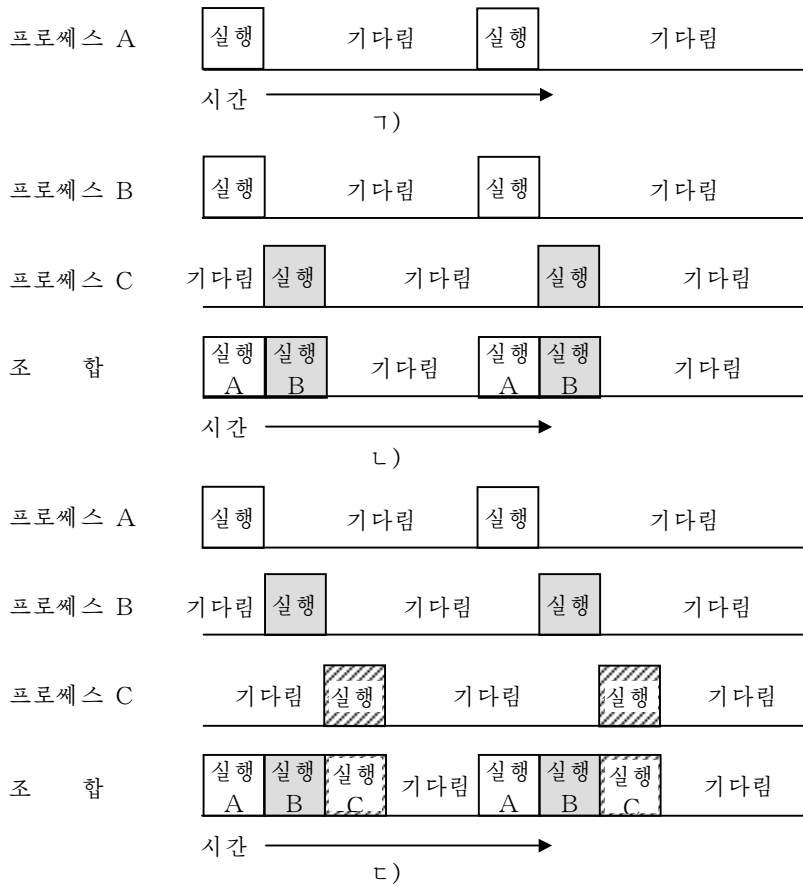



그림 2-5. 다중프로그람처리실례
ㄱ-단일프로그람처리, ㄴ-두개의 프로그람을 가진 다중프로그람처리
ㄷ-세개의 프로그람을 가진 다중프로그람처리

다중프로그람식일괄체계

단순일괄조작체계가 자동적으로 일감순서를 작성한다고 하여도 처리기는 흔히 휴식하게 된다. 문제는 입출력장치가 처리기에 비하여 뜨다는것이다. 그림 2-4에서는 구체적인 계산실례를 보여 주고 있다. 계산은 레코드당 평균 100개의 기계명령을 가지고 있는 레코드파일을 처리하는 프로그람과 관련한것이다. 이 실례에서 콤퓨터는 96%이상의 시간을 입출력장치가 파일과 자료이송을 끝낼 때까지 기다리는데 소비하고 있다. 그림 2-5 ㄱ는 이 상태를 보여 주고 있다. 그림에는 단일프로그람작성법으로 작성한 한개의 프로그람이 있다. 처리기는 입출력명령에 이를 때까지 일정한 시간을 집행하는데 소비한다. 그 다음 처리기는 입출력명령이 결속될 때까지 더 전진하지 않고 기다려야 한다.

이러한 비효률성은 필요 없다. 조작체계감시기와 사용자프로그람을 유지하자면 충분한 기억기가 있어야 한다. 조작체계와 두개의 사용자프로그람을 보관하기 위한 공간이 있다고 하자. 이제는 한 일감이 입출력장치의 기다림을 요구할 때 처리기가 다른 일감에로 절환할수 있다. 이것은 입출력을 기다리지 않는것과 류사하다(그림 2-5 ㄴ). 게다가 세개, 네개 기억기를 확장할수도 있다(그림 2-5 ㄷ). 이러한 처리를 **다중프로그람처리** 또

는 **다중과제처리**라고 한다. 이것이 현대 조작체계의 중심주제이다.

다중프로그람처리의 우점을 설명하기 위하여 간단한 실례를 들자. 256K단어의 기억기(조작체계가 쓰지 않는), 디스크, 말단, 인쇄기를 가진 콤퓨터를 고찰하자. 표 2-1에 제시된 속성들을 가지는 세개의 프로그람, 일감 1, 2, 3이 같은 시간에 집행될것을 요구하고 있다. 일감 2와 3에서는 최소한의 처리를 요구하며 일감 3은 디스크와 인쇄기를 련속적으로 사용한다고 가정한다. 표준일괄환경에서 일감들은 순차적으로 집행된다. 그러므로 일감 1은 5min동안에 완료되고 일감 2는 5min이 지나갈 때까지 기다려야 하며 그후 15min동안에 완료된다. 일감 3은 20min후에 시작하여 초기시간부터 30min만에 완료된다. 평균자원사용률, 처리능력, 응답시간을 표 2-2의 단일프로그람방식렬에 보여 주었다. 장치별 사용률을 그림 2-6에 보여 주고 있다. 평균 30min시간주기가 요구된다고 할 때 모든 자원들에서의 사용률이 낮다.

다중프로그람방식의 조작체계밑에서 일감들이 병행하여 실행된다고 가정하자. 일감들사이에서 자원경쟁이 적으므로 세개의 일감은 모두 콤퓨터내의 다른것들과 공존하면서 거의 최소시간내에 실행될수 있다(일감 2와 3이 입력과 출력조작을 하는데 충분한 처리기시간을 할당 받는다고 가정한다면). 일감 1은 완료하는데 5min을 요구하지만 그 시간에 일감 2는 3분의 1을 끝내며 일감 3은 절반을 끝내게 될것이다. 이 세가지 일감은 모두 15min에 완결될것이다. 이러한 개선정도는 그림 2-6 ㄴ에 보여 준 기둥도표로부터 얻어 진 표 2-2의 다중프로그람처리렬을 고찰해 보면 명백하다.

표 2-1. 견본프로그람집행속성

	일감 1	**일감 2**	**일감 3**
일감형태	대량계산	대량입출력	대량입출력
지속시간	5min	15min	10min
필요한 기억기량	50k	100k	80k
디스크요구?	없음	없음	있음
말단요구?	없음	있음	없음
인쇄기요구?	없음	없음	있음

단순일괄체계와 마찬가지로 다중프로그람식일괄체계는 일정한 콤퓨터장치특성에 의거하여야 한다. 다중프로그람처리에서 가장 주목할만한 특징은 입출력이 수행되는 동안 다른 일감을 계속 집행할수 있는것이다. 입출력조작이 완료되면 처리기는 조작체계에 있는 새치기조종프로그람에로 조종이 넘어 가게 된다. 그러면 조작체계는 조종을 또다른 일감에 넘긴다.

다중프로그람처리조작체계들은 단일프로그람 즉 **단일프로그람처리**에 비하여 상당히 복잡하게 얽힌 체계들이다. 실행할 준비가 되여 있는 몇가지 일감들을 가지자면 일정한 형의 **기억기관리**가 있어야 하며 주기억기내에 일감들이 유지되여 있어야 한다. 몇가지 일감들이 실행할 준비가 되면 처리기는 어느것을 실행하겠는가를 결정해야 하는데 이것은 일정한 일정작성알고리듬을 요구한다. 이 장에서 후에 이 개념들을 설명한다.

시분할체계

다중프로그람처리를 사용하면 일괄처리가 아주 효과적일수 있다. 그러나 많은 일감들에서 사용자가 직접 콤퓨터와 대화하는 방식을 요구한다. 실제상 트랜잭션처리와 같은 일감들에서 대화방식이 본질적인것으로 되고 있다.

오늘날 대화형계산기 등에 대한 요구는 전용극소형콤퓨터를 사용하여 만족시킬수 있다. 1960년대에는 대부분의 극소형콤퓨터들이 크고 비쌌기때문에 그런 선택을 할수 없었다. 그대신 시분할방법이 개발되였다.

다중프로그람처리는 처리기로 하여금 다중일괄일감들을 한번에 조종할수 있게 하는 것과 마찬가지로 다중프로그람처리를 다중대화형일감들을 조종하는데 사용할수 있다. 후자의 경우에 사용되는 수법을 시분할이라고 한다. 그것은 처리기의 시간이 다중사용자들속에서 공유되기때문이다. 시분할체계에서 다중사용자들은 말단들을 통하여 체계에 동시에 접근하며 조작체계는 짧고 무리적인 또는 량자적인 계산을 진행하는 방법으로 매개 사용자프로그람의 집행을 교차처리한다. 따라서 n개의 사용자가 한번에 봉사 받을것을 요청한다면 매 사용자는 조작체계의 간접소비시간을 고려하지 않는다면 유효콤퓨터속도의 평균 1/n동안만 봉사 받는것으로 된다. 그러나 사람의 반응속도가 상대적으로 낮다고 하면 합리적으로 설계된 체계의 응답시간은 전용콤퓨터의 응답시간과 비교할 정도로 된다.

일괄처리와 시분할처리는 모두 다중프로그람처리를 사용한다. 주요차이는 표 2-3에 제시되여 있다.

처음으로 개발된 시분할체계들중의 한개가 호환성 있는 시분할체계였다(CTSS)[CORB62]. 이것은 MAC(Machine-Aided Cognition 또는 Multiple-Access Computers)계획으로 알려 진 한 그루빠가 마샤츄세츠공과대학(MIT)에서 개발하였다. 이 체계는 1961년에 IBM 709용으로 처음 개발되였으며 후에 IBM 7094에 이식되였다.

후에 나온 체계들에 비해 볼 때 CTSS는 아주 원시적이였다. 그 체계는 32,000개의 36bit단어의 용량을 가진 기억기를 장비한 기계상에서 단어 5,000개를 차지하는 상주감시기를 사용하여 실행하였다. 대화형사용자에게 조종이 할당되면 사용자프로그람과 자료는 주기억기의 나머지 27,000개 단어에 적재되였다. 프로그람은 적재되여 항상 5,000번째 단어위치에서 시작하였다. 이것은 감시기와 기억기관리를 둘다 간단히 할수 있게 하

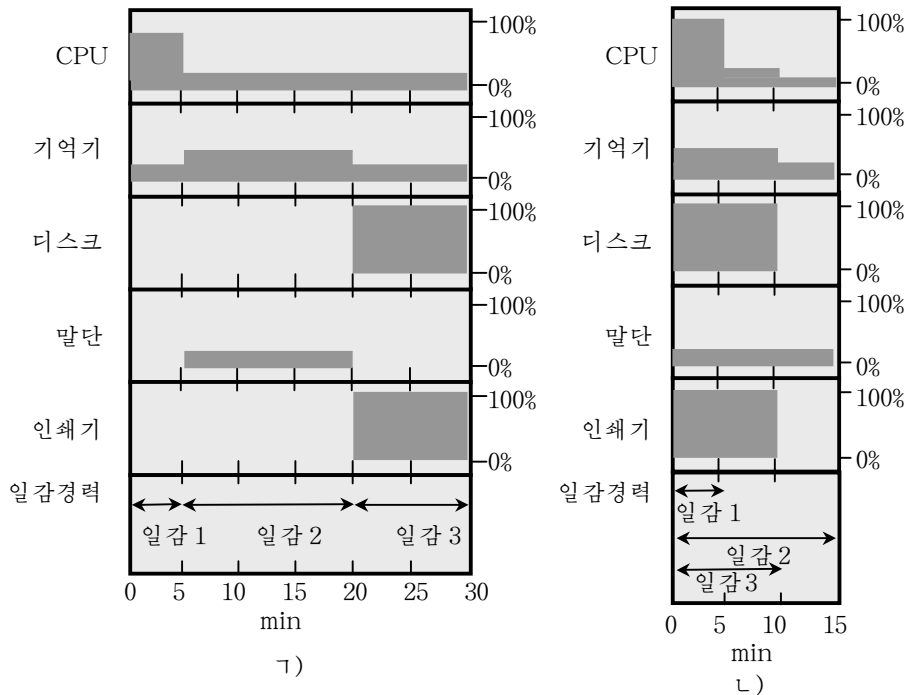



그림 2-6. 사용률도표: ㄱ-단일프로그람처리, ㄴ-다중프로그람처리

표 2-2. 자원사용률에 대한 다중프로그람처리의 효과

	단일프로그람처리	다중프로그람처리
처리기사용	32%	43%
기억기사용	30%	63%
디스크사용	33%	67%
인쇄기사용	33%	67%
경과시간	30min	15min
처리능력속도	6일감/h	12일감/h
평균응답시간	18min	10min

였다. 체계박자는 대략 0.2s에 한번의 속도로 새치기를 발생시켰다. 매개 박자새치기에서 조작체계는 조종을 회복하고 다른 사용자에게 처리기를 할당할수 있었다. 이렇게 하여 정규적인 시간간격으로 현재 사용자는 선취되고 다른 사용자가 적재된다. 원래의 사용자 프로그람상태를 후에 회복할수 있도록 보존하기 위하여 원래 사용자프로그람과 자료를 디스크에 써넣은 다음 새로운 프로그람과 자료를 읽어 들인다. 후에 원래 사용자프로그람코드와 자료는 자기 차례가 될 때 주기억기로부터 회복된다.

디스크정보흐름를 최소화하기 위하여 들어 오는 프로그람이 덧씌여 질 때에만 사용자프로그람을 디스크에 써넣는다. 이 원리를 그림 2-7에서 설명하고 있다. 다음과 같은 기억기요구를 가진 4개의 대화형사용자가 있다고 가정하자.

- 일감 1: 15,000
- 일감 2: 20,000

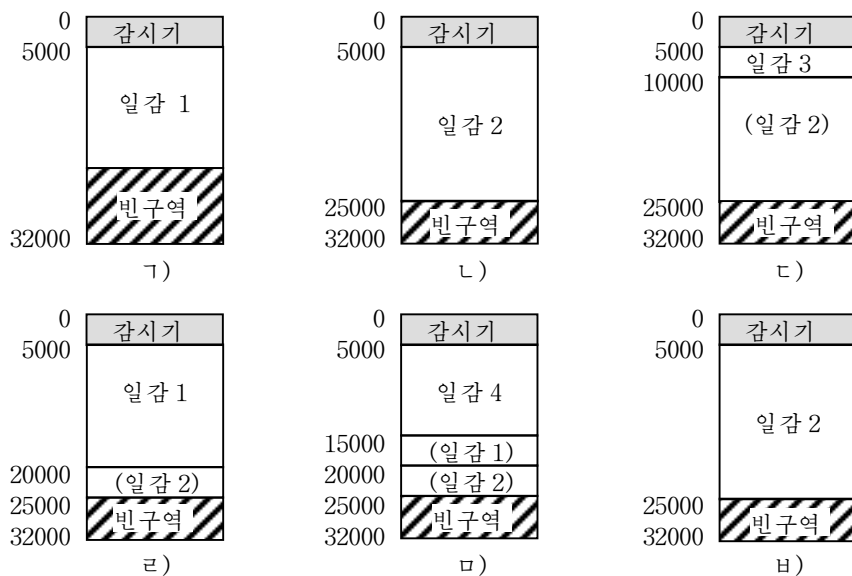



그림 2-7. CTSS 조작

표 2-3. 일괄다중프로그람처리 대 시분할처리

	일괄다중프로그람처리	시분할처리
기본목적	처리기사용을 최대로 한다.	응답시간을 최소화한다.
조작체계에 대한 지령문원천	일감을 제공하는 일감조종언어지령들	말단에서 들어 온 지령들

- 일감 3: 5000
- 일감 4: 10,000

처음에 감시기는 일감 1을 적재하고 조종을 그것에 이행한다(ㄱ). 감시기는 조종을 일감 2에 이행시킬것을 결정한다. 그것은 일감 2가 일감 1보다 더 많은 기억기를 요구하며 일감 1을 우선 외부쓰기해야 일감 2를 적재시킬수 있기때문이다(ㄴ). 다음으로 일감 3이 적재되여 실행된다. 그러나 일감 3은 일감 2보다 작기때문에 일감 2의 일부를 기억기에 남겨 둘수 있으며 디스크쓰기시간을 줄일수 있다(ㄷ). 그후 감시기는 조종을 일감 1에 다시 이행시키기로 결정한다. 일감 2의 추가부분은 일감 1이 기억기에 다시 적재될 때 외부쓰기되여야 한다(ㄹ). 일감 4가 적재될 때 일감의 일부와 일감 2의 일부는 기억기에 남아 있다(ㅁ). 이 시점에서 일감 1이든가 일감 2가 능동화된다면 부분적재만 요구될것이다. 이 실례에서 다음번에 실행해야 할것은 일감 2이다. 이것은 일감 4와 일감 1의 나머지부분을 외부쓰기하고 일감 2의 빠진 부분을 읽어 들일것을 요구한다.

CTSS방법은 오늘의 시분할방법에 비하여 원시적으로 동작하였다. 그것은 극히 단순하였고 감시기의 크기를 최소화하였다. 일감이 항상 주기억기의 같은 위치에 적재되였기때문에 적재시 치환기능이 전혀 요구되지 않았다(후에 설명한다.). 단지 필요한것을 외부쓰기하는 수법으로 디스크동작을 최소화하는것이였다. 7094상에서 실행할 때 CTSS는 최대 32명의 사용자를 지원하였다.

시분할처리와 다중프로그람처리는 조작체계에 많은 문제들을 제기하였다. 다중일감이 기억기안에 있다면 다른것들끼리 서로 간섭하는데로부터 실례로 다른것들의 자료를 수정시키지 못하도록 보호하여야 한다. 다중대화형사용자들에 대하여서는 권한을 가진 사용자만이 특정한 파일을 호출할수 있도록 파일체계를 보호해야 한다. 인쇄기와 대용량기억장치들과 같은 자원을 사용하는데서 발생되는 경쟁을 조종해야 한다. 이것과 다른 문제들의 해결방도에 대하여 책전반에서 론의할것이다.

제 3 절. 해결한 주요내용

조작체계는 지금까지 개발된 가장 복잡한 프로그람들중에 속한다. 이것은 곤난을 극복하려는 도전 그리고 일부 경우에 편리성, 효률성, 발전성에 대한 대상들사이의 경쟁을 반영하고 있다. [DENN80a]에서는 조작체계개발에서 5가지의 주요리론적전진이 있었다는것을 강조하고 있다. 즉

- 프로쎄스
- 기억기관리
- 정보의 보호와 보안
- 일정작성과 자원관리
- 체계구조

매개 전진은 어려운 실천적문제들을 해결하기 위하여 개발된 원리들이나 추상적개념

들로 특징 지어 진다. 이 5개의 령역은 현대 조작체계들의 주요설계와 실물들에서 다 같이 기초로 되고 있다. 이 절에서 5개 령역을 고찰해 보는것은 본문의 나머지부분들을 개괄하는데 도움을 주게 된다.

프로쎄스

프로쎄스에 대한 개념은 조작체계의 구조에서 기본개념이다. 이 용어는 1960년대에 Multics의 설계자들이 처음으로 사용하였다[DALE68]. 그것은 일감보다 더 일반적인 용어이다. 다음과 같은것들을 포함하여 용어 프로쎄스에 대한 정의들이 있다.

- 집행상태에 있는 프로그람
- 콤퓨터에서 실행하는 프로그람의 구체례
- 처리기에 할당할수 있고 처리기에서 집행할수 있는 구체례
- 단일하고 순차적인 집행스레드, 현 상태, 련관된 체계자원의 모임으로 특징 지어지는 동작단위

이 개념들은 앞으로 가면서 점차 명백해 질것이다.

콤퓨터체계개발의 세가지 방향인 다중프로그람방식의 일괄조작, 시분할, 실시간트랜잭션체계는 박자화와 동기화의 문제를 발생시켰는데 그것은 프로쎄스의 개념을 세우는데 기여하였다. 이미 본바와 같이 다중프로그람방식은 기억장치들을 포함하여 처리기와 입출력장치들이 동시에 동작하도록 설계하여 최대효률을 달성하도록 하였다. 주요한 수법은 입출력트랜잭션의 완료를 표시하는 신호에 응답하여 주기억기안에 있는 여러가지 프로그람들에 대하여 처리기를 절환하도록 하는것이다.

개발의 두번째 방향은 일반목적시분할이였다. 여기서 주되는 설계목표는 개별적인 사용자의 요구에 응답하는데 있으며 또한 비용상 견지에서 많은 사용자를 동시에 지원할수 있게 하는데 있다. 사용자의 반응시간이 상대적으로 느리므로 이 목표는 호환성을 가진다. 실례로 대체로 사용자가 처리시간을 분당 평균 2s 요구한다면 30명에 가까운 사용자를 간섭이 없이 같은 체계에 공유할수 있을것이다. 물론 조작체계의 간접소비시간은 고려해야 한다.

개발의 다른 중요한 방향은 실시간트랜잭션처리체계이다. 이 경우 많은 사용자가 자료기지에 문의하거나 갱신을 하려고 한다. 실례로 정기항공로예약체계를 들수 있다. 트랜잭션처리체계와 시분할체계사이의 주요차이는 전자는 한개 또는 몇개의 응용으로 제한되지만 반면에 시분할체계의 사용자들은 프로그람개발, 일감집행과 여러가지 응용프로그람의 사용에 참가할수 있다는것이다. 두 경우에 모두 체계응답시간이 가장 중요한 지표로 된다.

초기에 다중프로그람처리방식과 다중사용자대화체계를 개발하는데서 체계프로그람작성자가 할수 있는 기본도구는 새치기였다. 어떤 일감의 동작은 입출력완료와 같은 정의된 사건의 발생으로 중단시킬수 있었다. 처리기는 일정한 문맥(실례로 프로그람계수기나 다른 등록기들의 내용)을 보관하여야 했고 새치기의 특성을 판정할 새치기조종루틴에로 이행하여 새치기를 처리한 다음 새치기된 일감이나 어떤 다른 일감을 가지고 사용자처리를 회복하였을것이다.

여러가지 동작을 조화시켜 주는 체계프로그람설계는 많은 어려운 문제들을 산생시켰다. 임의의 한 시각에 진행중인 많은 일감들은 그 매개가 순차에 따라 수행해야 할 많은 단계들을 포함하고 있었으며 사건들의 순차에 대한 모든 조합을 분석하는것은 불가능하였다. 동작의 일치성과 협동에 대한 체계적인 의미들을 무시하고 프로그람작성자는 조작체계가 조종해야 할 환경을 자기식으로 리해한 기초우에서 특별한 방법에 매달리게 되였다. 이 결과 어떤 상대적으로 드문 동작순차들이 발생할 때에만 그 영향을 관찰할수 있

는 미묘한 프로그람작성오유들을 극복하기 힘들었다. 이 오유들은 응용프로그람오유와 하드웨어오유들을 구별해야 하였기때문에 진단하기 힘들었다. 오유가 검출되였다 하여도 원인을 판정하기 힘들었다. 왜냐하면 오유가 나타나는 정밀한 조건들을 다시 발생시키는 것이 매우 힘들었기때문이다. 일반적인 용어들로서 그러한 오유의 기본원인에는 4가지가 있다[DENN 80a]. 즉

- **부적당한 동기화 :** 이것은 흔히 체계내에서 다른 사건을 기다리면서 어떤 루틴을 중지시켜야 하는 경우이다. 실례로 입출력읽기를 시작하는 프로그람은 착수하기전에 완충기에서 자료를 불러 낼수 있을 때까지 기다려야 한다. 그런 경우에 일부 다른 루틴들로부터 어떤 신호를 요구하게 된다. 신호를 내보내는 기구를 부정확하게 설계하면 수신되는 신호를 잃어 버리거나 중복시키는 결과를 초래할수 있다.
- **고장호상배제 :** 이것은 흔히 한개이상의 사용자나 프로그람이 공유된 자원을 같은 시간에 사용하려고 하는 경우이다. 실례로 정기항공로예약체계에서 두 사용자가 자료기지를 읽으려고 할수 있다. 만일 좌석이 비여 있으면 예약하도록 자료기지를 갱신한다. 이 접근들을 조종하지 않으면 오유가 발생할수 있다. 일정한 종류의 호상배제기구가 있어서 한번에 한개의 루틴만이 일부 자료와 거래할수 있도록 허가해야 한다. 이러한 호상배제의 실현은 확인하기가 힘들다. 그것은 가능한 모든 사건순서하에서만 정확하기때문이다.
- **불명확한 프로그람조작 :** 특정한 프로그람의 결과들은 보통 프로그람의 입력에만 관계되며 공유된 체계내의 다른 프로그람들의 동작에는 관계되지 않는다. 그러나 프로그람들이 기억기를 공유하고 처리기가 그것들의 집행에 끼우게 되면 예측할수 없는 방식으로 공통기억기구역에 덧쓰기가 일어 나면서 서로 간섭할수 있다. 그러므로 여러가지 프로그람을 일정작성하는 순서가 특정한 프로그람의 결과에 영향을 줄수 있다.
- **교착 :** 둘이상의 프로그람이 서로 기다림상태에 들어가는 경우가 있을수 있다. 실례로 두개의 프로그람이 어떤 조작을 수행하기 위하여(실례로 테프에 디스크의 내용을 복사하기 위하여) 두개의 입출력장치들을 각각 요구할수 있다. 프로그람중의 하나가 두 장치중의 한개에 대한 조종을 차지하였고 다른 프로그람은 다른 장치에 대한 조종을 차지하였다. 각자는 다른 프로그람이 희망하는 자원을 해방시킬것을 기다리고 있다. 이러한 교착은 자원배정과 해방의 우연적인 시간맞물림에 의존할수 있다.

이 문제를 처리하는데 필요한것은 처리기에서 집행하는 여러가지 프로그람들을 관리하고 조종하는 체계적인 방법이다. 프로쎄스에 대한 개념은 그 기초를 준다. 프로쎄스가 세개의 구성성분들로 구성된다고 생각할수 있다.

- 집행가능한 프로그람
- 프로그람이 요구하는 련관된 자료(변수, 작업공간, 완충기 등)
- 프로그람의 집행문맥

이 마지막요소가 본질적이다. **집행문맥** 또는 **프로쎄스상태**는 조작체계가 프로쎄스를 관리하는데 필요하며 처리기가 프로쎄스를 완전히 집행하는데 필요한 모든 정보를 포함한다. 문맥에는 프로그람계수기와 그리고 자료등록기들과 같은 여러가지 처리기등록기들의 내용이 포함된다. 그것은 또한 처리의 우선권과 처리기가 특정한 입출력사건의 완료를 기다리고 있는가 아닌가 하는 조작체계에 대한 사용정보를 포함한다.

그림 2-8은 프로쎄스들을 실현할수 있는 방법을 가리켜 주고 있다. 두개의 프로쎄스 A와 B가 주기억기의 일부 구역들에 있다. 즉 기억기의 블로크가 매개 프로쎄스에 할당되여 있으며 그것은 프로그람, 자료와 문맥정보를 포함하고 있다. 매개 프로쎄스는 조작체계가 만들고 유지하고 있는 프로쎄스목록에 기록되여 있다. 프로쎄스목록은 매개 프로쎄스에 한개의 입구를 포함하고 있는데 그것은 프로쎄스를 포함하고 있는 기억기블로크의 위치에 대한 지시기이다. 입구는 또한 프로쎄스의 일부 또는 모든 집행문맥을 포함하고 있다. 집행문맥의 나머지는 프로쎄스자체와 함께 기억되여 있다. 프로쎄

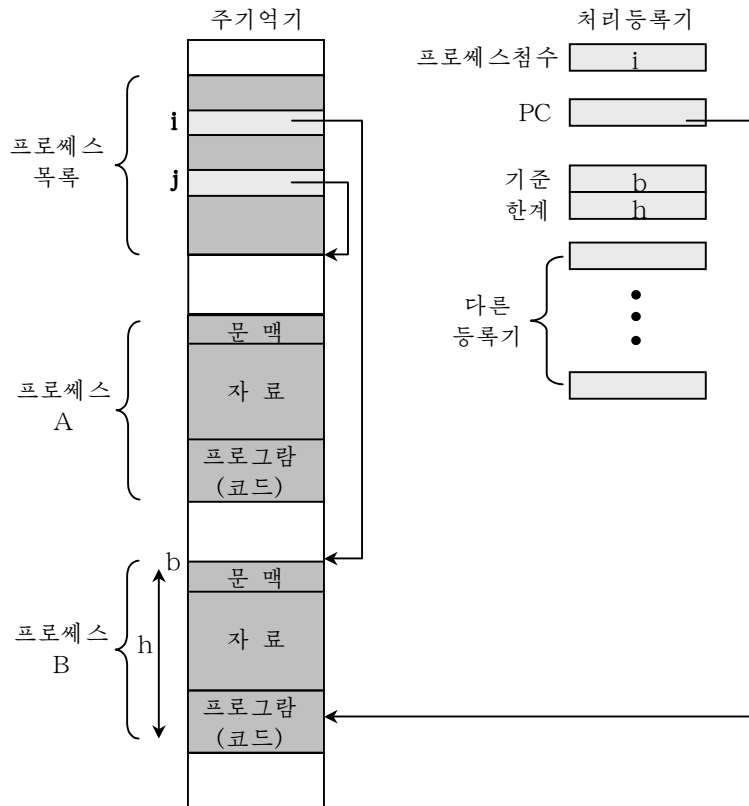



그림 2-8. 대표적인 프로쎄스의 실현

스의 첨수등록기는 현재 처리기를 조종하는 프로쎄스와 프로쎄스목록에 들어 가는 첨수를 포함하고 있다. 프로그람계수기는 집행하게 될 프로쎄스에서 다음명령의 주소를 가리킨다. 기준등록기와 한계등록기는 처리기가 차지한 기억기구역을 명시한다. 기준등록기는 기억기구역의 시작주소이며 한계등록기는 구역의 크기(바이트 또는 단어를 단위로)이다. 프로그람계수기와 모든 자료참조는 기준등록기에 대해 상대적으로 해석되며 한계등록기내의 값을 넘지 말아야 한다. 이것은 프로쎄스사이의 간섭을 막아 준다.

그림 2-8에서 프로쎄스첨수등록기는 프로쎄스 B가 집행되고 있다는것을 보여 준다. 프로쎄스 A는 이미 집행하고 있었지만 일시 새치기당하였다. A의 새치기순간에 모든 등록기들의 내용은 자기의 집행문맥에 기록되였다. 후에 조작체계는 프로쎄스절환을 수행할수 있으며 프로쎄스 A의 문맥을 회복할수 있다. 프로쎄스절환은 B의 문맥기억과 A의 문맥회복으로 구성되여 있다. 프로그람계수기가 A의 프로그람구역을 가리키는 값으로

적재될 때 A는 자동적으로 집행을 회복한다.

이와 같이 프로쎄스는 자료구조로 실현된다. 프로쎄스는 집행중이든가 집행기다림중에 있을수 있다. 임의의 순간에 프로쎄스의 완전한 **상태**는 그의 문맥에 포함되여 있다. 이 구조는 프로쎄스들속에서 일치성과 협동동작을 담보하는 강력한 수법을 개발할수 있게 한다. 기능을 지원하는데 요구되는 임의의 새로운 정보를 포함하도록 문맥을 확장함으로써 새로운 기능을 설계하여 조작체계에 병합시킬수 있다(실례로 우선권). 이 책전반에서는 많은 실례를 보게 되는데 여기서는 프로쎄스처리구조를 사용하여 다중프로그람작성과 자원공유에서 제기되는 문제들을 해결하고 있다.

기억기관리

모듈식프로그람작성과 자료를 유연하게 사용할수 있는 계산환경에 의하여 사용자들의 요구를 더 잘 만족시킬수 있다. 체계관리자들은 기억기배정을 효률적이며 정연하게 조종할것을 요구한다. 이 요구를 만족시키자면 조작체계는 5가지 기본적인 기억기관리기능을 가지고 있어야 한다.

- **프로쎄스고립 :** 조작체계는 자료와 기억기들이 서로 간섭하지 않도록 독립적인 프로쎄스들을 보호하여야 한다.
- **자동배정과 관리 :** 프로그람들은 기억기계층구조를 요구에 따라 동적으로 배정받게 된다. 배정은 프로그람작성자에게 투명적이여야 한다. 이와 같이 하여 프로그람작성자는 기억기한계를 우려하지 않아도 되며 조작체계는 필요할 때에만 일감들에 기억기를 배정함으로써 효률을 높일수 있다.
- **모듈식프로그람작성의 지원 :** 프로그람작성자들은 프로그람모듈들을 정의하고 창조하며 모듈들의 크기를 동적으로 변경시킬수 있다.
- **보호와 접근조종 :** 기억기계층구조의 임의의 준위에서 기억기를 공유하는것은 한 프로그람이 다른 프로그람의 기억기공간을 주소지정할수 있는 가능성을 준다. 이것은 특정한 응용프로그람이 공유를 요구할 때 필요하다. 다른 때에는 프로그람들의 완전성과 조작체계기억기의 부분들에 접근할수 있게 하여야 한다.
- **장기기억 :** 많은 응용프로그람들은 콤퓨터전원이 꺼진 다음에 확장된 시간주기동안 정보를 기억하기 위한 수단을 요구한다.

대체로 조작체계들은 가상기억기와 파일체계기능들을 사용하여 이 요구를 만족시킨다. 파일체계는 장기기억을 실현하는데 이때 정보는 이름 붙은 대상들과 호출된 파일들로 기억된다. 파일은 프로그람작성자에게 편리한 개념으로서 접근조종과 조작체계보호의 유용한 단위로 된다.

가상기억기는 프로그람들이 물리적으로 가능한 주기억기의 량을 고려하지 않고 론리적인 견지에서 기억기를 주소화할수 있게 하는 장치이다. 가상기억기는 다중사용자일감들을 주기억기안에 동시에 상주시켜 한개의 프로쎄스를 2차기억기에 써내고 다음 프로쎄스를 읽어 들이는 동안 련속적인 프로쎄스집행사이에 일어 나는 새치기를 없앨데 대한 요구를 만족시키기 위하여 만들어 냈다. 프로쎄스들은 크기가 다르므로 처리기가 많은 프로쎄스들속에서 절환된다면 그것들을 밀집하여 주기억기에 넣기 곤난하다. 이런데로부터 페지화체계를 도입하였는데 이것은 프로쎄스들을 페지라는 많은 고정크기의 블로크들로 만든다. 프로그람은 페지번호와 페지안에서의 편위로 구성되는 **가상주소**를 사용하여 단어를 참조한다. 프로쎄스의 매개 페지를 주기억기의 임의의 곳에 배치할수 있다. 페지화체계는 프로그람에 사용된 가상주소와 **실제주소** 즉 주기억기의 물리적주소사이에서의

동적배치를 할수 있게 한다.

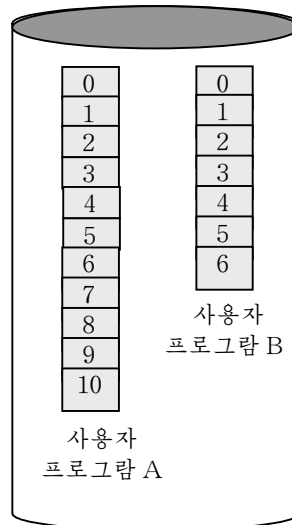
주소사영하드웨어를 사용하는 경우에 다음의 론리적단계는 프로쎄스의 모든 페지들을 주기억기에 동시에 상주시킬데 대한 요구를 없애는것이였다. 프로쎄스의 모든 페지들은 디스크상에 보관되여 있다. 어떤 프로쎄스가 집행중에 있을 때 그의 일부 페지들은 주기억기에 있다. 만일 참조가 주기억기에 없는 페지에서 이루어 진다면 기억기관리장치는 이것을 검출하고 빠진 페지가 적재되도록 한다. 이러한 도식을 가상기억기라고 하며 그림 2-9에 보여 주고 있다.

처리기하드웨어는 조작체계와 함께 사용자가 가상기억기에 접근하는 《가상처리기》를 가질수 있게 한다. 가상기억기는 선형주소공간 또는 토막들의 집합일수 있다. 토막은 린접하는 주소를 가진 가변길이블로크들이다. 어느 경우든 프로그람작성언어명령은 가상기억기에서 프로그람과 자료의 위치들을 참조할수 있다. 매개 프로쎄스들에 단일하고 겹치지 않는 가상주소를 주어 프로쎄스들을 고립시킬수 있다. 두개의 가상기억기공간의 일부분을 겹치게 하여 기억기공유를 실현할수 있다. 파일들은 장기기억기에 보관되여 있다. 파일들과 파일의 일부분들을 프로그람들이 다룰수 있도록 가상기억기에 복사시킬수 있다.

그림 2-10은 가상기억기조작에서 주소화관계를 강조하고 있다. 기억기는 직접주소화가능한(기계명령들에 의하여) 주기억기와 블로크들을 주기억기에 적재시켜 간접적으로 접근하는 저속보조기억기로 구성되여 있다. 처리기와 기억기사이에 주소변환하드웨어(주소사영기)가 들어 간다. 프로그람들은 가상주소로 위치를 참조하는데 가상주소는 실제주기억기주소로 사영된다. 만일 실제기억기주소가 아니라 가상주소로 참조한다면 실제기억기내용의 일부가 보조기억기에로 바뀌여 나가고 요구하는 자료블로크가 바뀌여 들어간다. 이 동작이 진행되는 동안에 주소참조를 발생한 프로쎄스는 중단시켜야 한다.설계

A.1			
	A.0	A.2	
	A.5		
B.0	B.1	B.2	B.3
		A.7	
	A.9		
		A.8	
B.4	B.5	B.6	

기억기

주기억기는 페지의 크기와 같은 고정길이의 많은 프레임들로 구성되여 있다. 프로그람을 실행하기 위하여 일부 페지 또는 모든 페지를 주기억기에 넣어야 한다.

디스크

2 차기억기(디스크)는 고정길이의 많은 페지들을 가지고 있다. 사용자프로그람은 몇개의 페지들로 되여 있다. 모든 프로그람과 조작체계의 페지들은 파일로서 디스크에 있다.

그림 2-9. 가상기억기의 개념

자의 과업은 주소변환기구를 개발하여 간접소비시간을 작게 하며 기억기준위들사이에서 정보흐름을 최소화하는 기억기배정방법을 확립하는것이다.

정보의 보호와 보안

시분할체계와 최근 콤퓨터망사용이 확대되면서 정보보호에 대한 관심이 높아 졌다. 어떤 방식에 관계되는 토막과제의 본질은 환경에 따라 크게 변화된다. 그러나 콤퓨터와 조작체계들에 만들어 넣을수 있는 일부 일반목적의 도구들이 있다. 그 도구들은 다양한 보호와 보안수법를 준다. 일반적으로는 콤퓨터체계와 거기에 기억된 정보들에 대한 접근을 조종하는 문제에 관심을 가진다.

보안과 보호에서의 많은 작업은 조작체계와 관련이 있는것으로서 세개의 범주로 묶어 볼수 있다. 즉

- **접근조종 :** 총적인 체계, 부분체계, 자료에 대한 정규적인 사용자접근과 체계에 있는 여러가지 자원과 개체들에 대한 프로쎄스접근과 관련된 조종이다.
- **정보흐름조종 :** 체계의 자료와 그것을 사용자들에게 분배하는 흐름을 조절한다.
- **인증 :** 접근 및 흐름조종기구들이 자기 명세대로 동작하며 그것들이 희망하는 보호와 보안방법을 실시하는가를 확증한다.

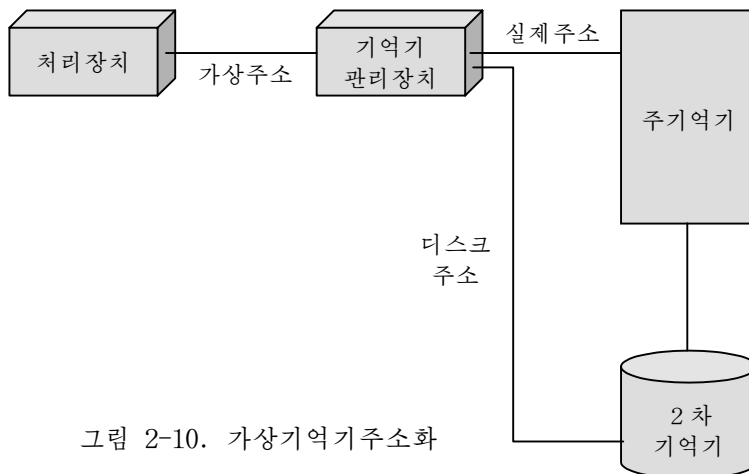



그림 2-10. 가상기억기주소화

일정작성과 자원관리

조작체계의 기본과제는 사용가능한 여러가지 자원들(주기억기공간, 입출력장치들, 처리기들)을 관리하는것이며 여러가지 능동프로쎄스들이 그것을 사용하는 일정을 작성하는 것이다. 임의의 자원배정과 일정작성방식은 세가지 인자들을 고려하여야 한다. 즉

- **공평성 :** 대체로 특정한 자원을 사용하려고 경쟁하는 모든 프로쎄스들이 그 자원에 거의 동일하게 그리고 공평하게 접근하도록 하려고 한다. 특히 같은 일감들 즉 류사한 요구를 가진 일감들에 대하여 그러하다.
- **차분응답성 :** 한편 조작체계는 서로 다른 봉사요구를 가진 각이한 부류의 일감들을 식별해 낼것을 요구한다. 조작체계는 총체적인 요구들을 만족시키도록 배정 및 일정작성을 결정하여야 한다. 조작체계는 또한 이 결정을 동적으로 진행하여

야 한다. 실례로 어떤 프로쎄스가 입출력장치를 기다린다면 조작체계는 그 프로쎄스가 가능한 집행을 빨리 끝내고 장치를 해방하여 다른 프로쎄스들의 요구를 만족시키도록 일정을 작성하려고 할수 있다.

- **효률** : 조작체계는 처리능력을 최대로 되게 하고 응답시간을 최소화하며 시분할인 경우 가능한 많은 사용자들의 편의를 도모하려고 시도해야 한다. 이 기준은 어느 정도 모순적이다. 특정한 환경에서 균형을 옳게 맞추는것이 조작체계연구의 현실적인 문제이다.

일정작성과 자원관리과제는 본질적으로 조작들에 대한 연구문제이며 련마된 수학적 결과들을 적용할수 있다. 더우기 체계동작을 측정하여 감시기가 조정할수 있게 하는것이 중요하다.

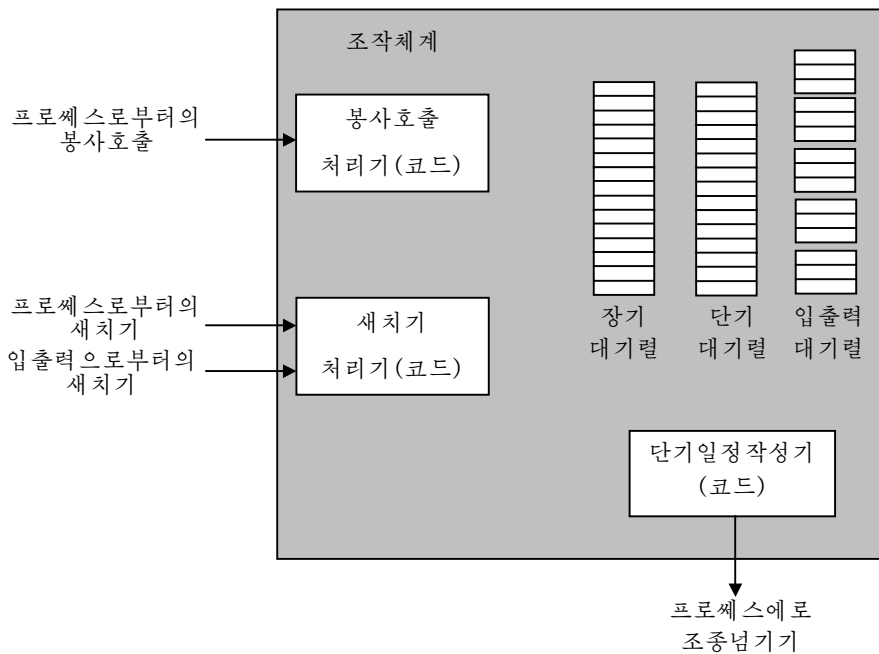



그림 2-11. 다중프로그람방식에서 조작체계의 기본요소들

그림 2-11은 다중프로그람처리환경에서 프로쎄스들의 일정작성에 포함된 조작체계의 기본요소들과 자원할당을 보여 주고 있다. 단순히 어떤 자원을 기다리는 프로쎄스들의 목록이다. 단기대기렬은 주기억기에 있는(또는 적어도 기본적인 최소부분이 주기억기에 있는) 그리고 실행할 준비가 되여 있는 프로쎄스들로 구성되여 있다. 이 프로쎄스들중에서 임의의 한개가 처리기를 다음번에 사용할수 있다. 그 한개를 골라 내는것이 바로 단기일정작성기 또는 배분프로그람의 임무로 되여 있다. 공통적인 전략은 대기렬안에 있는 매개 프로쎄스들에 차례로 일정한 시간이 차례지게 하는것이다. 이것을 **순환법**이라고 한다. 역시 우선권준위들도 사용할수 있다.

장기대기렬은 처리기를 사용하려고 기다리는 새로운 일감들의 목록이다. 조작체계는 프로쎄스를 장기대기렬에서 단기대기렬에 이송시킴으로써 체계에 일감들을 추가한다. 이 경우 들어 오는 프로쎄스에 주기억기의 일부를 배정하여야 한다. 이렇게 함으로써 조작

체계는 너무 많은 프로쎄스들을 체계에 허락해 주어 기억기나 처리시간에서 문제가 생기지 않도록 담보하여야 한다. 매개 입출력장치에 입출력대기렬이 있다. 매개 장치를 사용하려고 기다리는 프로쎄스들은 모두 장치들의 대기렬에 줄 서게 된다. 그때 조작체계는 어느 프로쎄스를 사용가능한 입출력장치들에 할당하겠는가를 결정해야 한다.

조작체계는 새치기가 발생하면 새치기조종기에서 처리기의 조종을 받는다. 프로쎄스는 봉사호출방법으로 입출력장치조종기와 같은 일정한 조작체계봉사를 특정하게 기동시킬수 있다. 이 경우 봉사호출조종기는 조작체계에 들어 가는 입구점으로 된다. 임의의 경우 일단 새치기나 봉사호출을 조종하면 단기일정작성기를 기동하여 집행하기 위한 프로쎄스를 골라 내게 한다.

우에서 말한것은 기능적인 설명이다. 즉 조작체계의 이 부분에 대한 세부와 모듈설계는 여러가지 체계들에서 각이할것이다. 조작체계에서 많은 연구와 개발노력은 공평성, 각이한 응답성, 예측가능성 및 효과성을 보장하는 기능들을 위한 알고리듬과 자료구조를 선택하는데 관심을 돌렸다.

체계구조

조작체계에 기능들이 점점 추가됨에 따라 그리고 그 밑에 놓이는 하드웨어가 보다 능력 있는 다방면적인것으로 되여 감에 따라 조작체계의 규모와 복잡성도 커졌다. 1963년에 MIT에서 연산에 도입된 CTSS는 대략 32,000개의 36bit단어를 가진 기억기로 구성하였다. 1년후에 IBM이 도입한 조작체계 OS/360에서는 기계명령이 백만개이상이였다. 1975년까지 MIT와 Bell연구소에서 개발한 Multics체계에서는 2천만개이상의 명령을 가지는 정도로 확대되였다. 최근에 보다 단순한 조작체계들은 보다 작은 체계용으로 도입하였지만 그 밑에 놓여 있는 하드웨어와 사용자의 요구가 높아 졌으므로 이 조작체계들은 불가피하게 복잡성을 더욱 증가시켰다. 따라서 오늘날의 UNIX은 1970년대초에 몇명의 재능 있는 프로그람작성자들이 만들어 넣은 체계들보다 훨씬 더 복잡하며 간단한 MS-DOS는 OS/2와 Windws2000의 풍부하고 복잡한 능력을 갖추게 하기 위한 방도를 주었다. 실례로 Windows NT 4.0에서는 천6백만행의 코드를 포함하고 있으며 Windows 2000에서는 그것이 2배이상으로 되고 있다.

충분한 기능을 갖춘 조작체계의 규모와 그것이 지정하는 과제의 난점은 모든 체계에 공통적인 세가지 문제를 초래하였다. 이것은 새 조작체계에 한한것이고 낡은 체계는 갱신된다. 첫째로, 조작체계가 배포에서 시간이 걸리는것이다. 둘째로, 체계들이 운영과정에 드러나서 그것을 수정하거나 재작업해야 하는 잠복오유를 가지고 있는것이다. 셋째로, 성능이 기대하던것보다 못하다는것이다.

조작체계의 복잡성을 처리하고 이 문제들을 극복하기 위하여 조작체계의 쏘프트웨어 구조에 오래동안 초점이 집중되여 왔다. 어떤 점들은 명백한것 같다. 쏘프트웨어는 모듈식으로 되여야 한다. 이것은 쏘프트웨어개발과정을 조직하며 오유를 진단하고 수정시키는 과제들을 제한하는데서 도움을 준다. 모듈은 서로 잘 정의된 대면부를 가져야 하며 대면부들은 될수록 간단해야 한다. 이것은 또한 프로그람작성과제를 쉽게 해준다. 그것은 역시 체계의 발전을 더 쉽게 할데 대한 과제를 해결할수 있게 한다. 모듈들사이에서 명백하고 최소인 대면부를 가지고 있으면 한개의 모듈을 다른 모듈에 최소의 영향을 주도록 변화시킬수 있다.

큰 조작체계들을 보면 백만내지 수천만행의 코드를 가지고 실행되며 모듈식프로그람 작성하나만으로 충분하다고 볼수 없게 되였다. 그대신에 계층구조적인 충들과 정보추상화에 대한 개념들을 사용하는 경향이 커지게 되였다. 현대조작체계의 계층구조는 그것들의 복잡성, 특징적인 시간척도, 추상화준위에 따라 자기의 기능을 분리시키고 있다. 이때

체계를 어떤 준위들의 계렬로 볼수 있다. 매개 준위는 조작체계가 요구하는 기능들의 련관된 부분모임을 형성한다. 그것은 보다 원시적인 기능을 수행하며 그 기능들의 세부를 교갑화하고 있는 다음의 보다 높은 층에 봉사한다. 리상적으로 준위들은 한 준위에서의 변화가 다른 준위들에서의 변화를 요구하지 않도록 정의되여야 한다. 이로부터 한개의 문제를 관리할수 있는 많은 부분문제들로 분해하였다.

일반적으로 보다 낮은 층들은 훨씬 짧은 시간척도를 취급한다. 조작체계의 일정한 부분들은 콤퓨터하드웨어와 직접 대화하여야 하며 여기서 사건들은 수십억분의 몇초와 같은 짧은 시간척도를 가질수 있다. 많은 과제들중 다른 한 극단에서 볼 때 조작체계의 일부는 사용자와 통신하는데 이때 사용자는 훨씬 느린 속도 즉 몇초에 한번 정도로 지령을 내보낸다. 준위모임들에 대한 사용은 이 환경에 훌륭히 적응하게 한다.

이 원리들을 적용하는 방법은 당시 조작체계들속에서 크게 차이난다. 그러나 여기에서 필요한것은 조작체계에 대한 개괄을 줄 목적으로 계층구조적인 조작체계모형을 보여주는것이다. [BROW84a]와 [DENN84]에서 제기한 모형은 어떤 특정한 조작체계에 대응하는것이 아니지만 유용하다. 이 모형은 표 2-4에서 정의해 주고 있으며 다음의 준위들로 구성되여 있다. 즉

- **1준위** : 전자회로로 구성되여 있으며 여기에 등록기, 기억세포, 론리문을 취급하는 객체들이 있다. 이 객체들에 정의된 조작들은 등록기지우기나 기억위치읽기와 같은 동작이다.
- **2준위** : 처리기의 명령모임이나 이 준위에서의 조작을 더하기, 덜기, 적재, 기억과 같은 기계어명령모임으로 하게 되여 있다.
- **3준위** : 수속이나 부분루틴과 호출/복귀조작에 대한 개념을 추가한다.

표 2-4. 조작체계설계의 계층구조

준위	이름	객체	실례조작
13	쉘	사용한 프로그람작성환경	쉘언어로 된 설명문
12	사용자프로쎄스	사용자프로쎄스	리탈, 삭제, 중단, 회복
11	등록부	등록부	창조, 파괴, 첨부, 떼여내기, 탐색, 목록
10	장치	인쇄기, 현시기, 건반과 같은 외부장치	열기, 닫기, 읽기, 쓰기
9	파일체계	파일	창조, 파괴, 열기, 닫기, 읽기, 쓰기
8	통신	흐름관	창조, 파괴, 열기, 닫기, 읽기, 쓰기
7	가상기억기	토막, 페지	읽기, 쓰기, 불러내기
6	국부2차기억	자료블로크, 장치통로	읽기, 쓰기, 배정, 해방
5	원시적프로쎄스	원시적프로쎄스들,신호기, 준비목록	중단, 회복, 기다림, 신호
4	새치기	새치기조종프로그람	기동, 마스크, 마스크해저, 재시도
3	수속	수속, 호출, 탄창, [illegible]시	표식탄창, 호출, 복귀
2	명령모임	평가탄창, 마이크로프로그람 해석기, 스칼라 및 [illegible]렬자료	적재, 기억, 더하기, 덜[illegible], 갈래
1	전자회로	등록기, 론리문, 모[illegible] 등	지우기, 이송, 능동, 보[illegible]

어두운 칸은 하드웨어를 표시한다.

- **4준위 :** 새치기를 도입하는데 이것은 처리기가 현재의 문맥을 보관하고 새치기조종루틴을 끌어 들이게 한다.

이 첫 4개 준위들은 조작체계부분이 아니라 처리기하드웨어를 구성한다. 그러나 일부 조작체계요소들은 새치기조종루틴과 같이 이 준위들에서 나타나기 시작한다. 조작체계에 접근하기 시작하며 다중프로그람처리와 관련된 개념들이 나타나는것이 바로 5준위이다.

- **5준위 :** 집행중에 있는 프로그람과 같이 프로쎄스에 대한 일반개념이 이 준위에서 도입된다. 다중프로쎄스를 지원하는 조작체계에 대한 기본요구는 프로쎄스를 중단하고 회복하는 능력을 포함하는것이다. 이것은 하드웨어등록기들을 보관하여 한 프로쎄스에서 다른 프로쎄스에로 집행을 절환시킬수 있다. 또한 프로쎄스들이 협동조작할것을 요구하면 어떤 동기화방법이 필요하다. 조작체계설계에서 하나의 간단한 수법이면서도 중요한 개념은 신호기인데 이것은 단순한 신호조작수법으로서 제5장에서 론의한다.
- **6준위 :** 콤퓨터의 2차기억장치를 취급한다. 이 준위에서 읽기/쓰기머리와 실제자료블로크이송을 포함하는 기능들이 발생한다. 6준위는 5준위에 의거하여 조작을 일정작성하고 조작완료프로쎄스의 요청을 통지한다. 보다 높은 준위들은 디스크의 필요한 자료주소와 련계되며 5준위의 장치구동을 위해 해당한 블로크에 대한 요청을 제기한다.
- **7준위 :** 프로쎄스들을 위한 론리주소공간을 창조한다. 이 준위는 주기억기와 2차기억기사이에서 이동시킬수 있는 블로크에 대한 가상주소공간을 조직한다. 세개의 방안을 공통적으로 사용한다. 즉 고정크기페지를 사용하는것, 가변길이토막을 사용하는것, 이 두가지를 다 사용하는것들이다. 필요한 블로크가 주기억기에 없으면 이 준위에서의 론리는 6준위로부터 블로크를 이송할것을 요청한다.

이 시점까지 조작체계는 단일처리기의 자원을 취급한다. 8준위부터 조작체계는 주변장치들과 가능하게는 망 그리고 망에 소속된 콤퓨터들과 같은 외부객체들을 취급한다. 이 웃준위에 있는 객체들은 론리적이며 이름 붙은 객체들로서 같은 콤퓨터상에 있거나 다중콤퓨터상에 있는 프로쎄스들사이에서 공유시킬수 있다.

- **8준위 :** 프로쎄스들사이에 정보와 통보문들의 통신을 취급한다. 한편 5준위는 프로쎄스들의 동기화를 보장할수 있게 하는 원시적신호기구를 주는데 이 준위는 더 풍부한 정보준위를 취급한다. 이 목적에서 가장 강력한것중의 하나가 흐름관인데 이것은 프로쎄스들사이에서 자료흐름을 위한 론리적인 통로로 된다. 흐름관은 한 프로쎄스로부터 나가는 출구와 다른 프로쎄스에로 들어 가는 입구로 정의된다. 그것은 또한 외부장치나 파일들을 프로쎄스에 련결하는데 사용할수 있다. 이 개념은 제6장에서 설명한다.
- **9준위 :** 이름을 가진 장기기억을 지원한다. 이 준위에서 2차기억기상의 자료는 추상적이며 가변길이를 가진 구체례들로 보인다. 이것은 6준위에서 자리길, 분구, 고정크기블로크에 의하여 하드웨어지향적으로 보는것과는 대조적이다.
- **10준위 :** 표준화된 대면부를 사용하여 외부장치들에 접근할수 있게 한다.
- **11준위 :** 체계의 자원과 객체들에 대한 외부 및 내부식별자사이에서 련계를 유지한다. 외부식별자는 프로그람이나 사용자가 사용할수 있는 이름이다. 내부식별자는 낮은 준위의 조작체계가 어떤 객체를 찾고 조종하는데 사용할수 있는 주소나 다른 지시기이다. 련관관계는 등록부에 유지되여 있다. 입구들은 외부/내부배치뿐

아니라 접근권한과 같은 특성도 포함하고 있다.

- **12준위 :** 프로쎄스들을 지원하는데 충분한 특징을 가진 기능을 준다. 이것은 5준위에서 제공하는것을 훨씬 초월한다. 5준위에서는 프로쎄스와 관련된 처리기등록기내용만이 유지되여 있지만 여기서는 그밖에 프로쎄스들을 배분하기 위한 론리가 있다. 12준위에서는 프로쎄스들을 질서 있게 관리하는데 필요한 모든 정보를 준다. 이것은 프로쎄스들에 대한 가상주소공간, 그것이 대화할수 있는 객체와 프로쎄스들의 목록과 대화의 제한조건, 창조할 때 프로쎄스에 넘겨 준 파라메터들과 조작체계가 프로쎄스를 조종할 때 사용할수 있는 프로쎄스의 어떤 다른 특성들을 포함하고 있다.
- **13준위:** 사용자에게 조작체계와의 대면부를 제공한다. 이것을 **쉘**이라고 하는데 그 리유는 이것이 사용자를 조작체계세부들로부터 분리시키고 조작체계를 단순히 봉사의 집합으로 볼수 있게 하기때문이다. 쉘은 사용자지령이나 일감조종명령들을 접수하고 이것들을 해석실행하며 필요에 따라 프로쎄스들을 창조하고 조종한다. 실례로 이 준위에서의 대면부는 도형방식으로 실행될수 있다. 말하자면 사용자에게 어떤 메뉴로 표시된 지령을 주며 화면과 같은 특정한 장치에로의 도형출력을 사용하여 결과들을 현시할수 있다.

조작체계에 대한 가설모형은 유용한 서술구조를 주며 실현안내틀로 봉사한다. 독자들은 이 책을 읽는 과정에 어떤 특정한 설계문제가 토의되는 경우 이 구조를 사용할수 있다.

제 4 절. 현대적인 조작체계의 특성

여러해를 거쳐 조작체계구조와 능력에서 원만한 발전이 이룩되였다. 그러나 최근년간에 새로운 조작체계와 현존조작체계의 판본에 몇가지 새로운 설계요소들을 도입하였는데 이것은 조작체계의 특성에서 중요한 변화를 일으키고 있다. 현대적인 조작체계들은 새로운 하드웨어개발과 새로운 응용프로그람에 대응한다. 주요한 하드웨어구동기들중에는 기계속도가 크게 증가된 다중처리기식기계들과 고속망설비들, 기억기들의 크기와 종류가 다양한 기억장치들이 있다. 응용분야를 보면 다매체응용, 인터네트와 Web접근 및 의뢰기/봉사기계산이 조작체계설계에 영향을 주었다.

조작체계에 대한 수요의 변화률은 현존구성방식의 변경과 강화가 아니라 조작체계를 새로운 방식으로 구성할것을 요구하고 있다. 실험실적 및 상업적조작체계들에서 모두 넓은 령역의 각이한 방법들과 설계요소들을 적용하려고 시도해 왔는데 많은 작업은 다음의 범주들에 속하는것이다. 즉

- 마이크로핵심부구성방식
- 다중스레드처리
- 대칭다중처리
- 분산조작체계
- 객체지향설계

대부분의 조작체계들은 최근까지 하나의 큰 **단일핵심부**로 특징 지을수 있다. 조작체계기능이라고 할 때에는 대부분이 이 큰 핵심부에서 제공되고 있다. 여기에는 일정작성 파일체계, 망화, 장치구동프로그람구체례, 기억기관리 및 기타가 포함된다. 대체로 단일핵심부는 단일프로쎄스로 실행되는데 여기서 모든 요소들은 같은 주소공간을 공유하고 있다. 이 **핵심부구성방식**에서는 몇개의 본질적인 기능만을 핵심부에 할당한다. 주소공간,

프로쎄스사이통신(IPC), 기본적인 일정작성을 포함하여 조작체계의 다른 봉사는 프로쎄스들에 의하여 제공되는데 이런 프로쎄스들을 때때로 봉사기라고 부른다. 이것은 사용자방식으로 실행되며 마이크로핵심부에 의하여 다른 응용프로그람과 같이 취급된다. 이 방법은 핵심부와 봉사기개발을 분리시킨다. 특정한 응용이나 환경요구에 맞게 봉사기를 주문하여 개발할수 있게 한다. 마이크로핵심부방법은 실현을 간소화해 주며 유연성을 제공하며 분산환경에 잘 어울린다. 본질상 마이크로핵심부는 국부 및 원격봉사기프로쎄스들과 같은 방법으로 대화하며 분산체계구성을 촉진시키고 있다.

다중스레드처리는 응용프로그람을 집행하는 프로쎄스를 병행하여 실행할수 있는 스레드들로 분할하는 수법이다. 이것을 다음과 같이 구별할수 있다. 즉

- **스레드 :** 배분할수 있는 작업단위이다. 이것은 처리기의 문맥(프로그람계수기와 탄창지시기를 포함하고 있다)과 그자체의 탄창자료구역(보조루틴의 갈래를 가능하게 한다.)을 포함하는데 스레드는 순차적으로 집행하며 처리기나 다른 스레드에로 넘어 갈수 있도록 새치기할수 있다.
- **프로쎄스 :** 한개 또는 그이상의 스레드모임으로서 관련 있는 체계자원이다(코드와 자료를 모두 포함하고 있는 기억기, 열린 파일, 장치들과 같은). 이것은 집행중에 있는 프로그람에 대한 개념과 거의 대응한다. 단순한 응용프로그람을 다중스레드들로 분할함으로써 프로그람작성자는 응용프로그람의 모듈과 전반에 대한 많은 조종을 해야 하며 응용프로그람과 관련된 사건들을 시간맞추기해야 한다.

다중스레드처리는 직렬화할것을 요구하지 않는 몇개의 본질적으로 독립적인 과제들을 수행하는 응용프로그람에서 쓸모가 있다. 실례로서 수많은 의뢰기요청들을 듣고 처리하는 자료기지봉사기를 들수 있다. 같은 프로쎄스에서 여러개의 스레드들을 실행할 때 스레드들사이의 앞뒤절환은 각이한 프로쎄스들사이에서 기본프로쎄스절환보다 더 작은 처리기간접소비시간을 가진다. 스레드들은 또한 조작체계핵심부의 일부분인 프로쎄스들을 구조화하는데서 쓸모가 있다. 이 내용은 다음장들에서 서술한다.

최근까지 사실상 모든 단일사용자형개인용콤퓨터들과 워크스테이션들은 단일한 범용의 극소형처리기를 내장하였다. 성능을 높일데 대한 요구에 따라 그리고 극소형처리기들의 비용이 계속 떨어 짐에 따라 판매자들은 다중극소형처리기를 가진 콤퓨터들을 도입하였다. 보다 큰 효률성과 믿음성을 달성하기 위한 한가지 수법은 **대칭다중처리**(SMP)를 사용하는것인데 이 용어는 콤퓨터하드웨어구성방식과 또한 그 구성방식을 반영하는 조작체계를 가리키는 말이다. 대칭형다중처리기를 다음과 같은 특성을 가진 독립형콤퓨터체계로 정의할수 있다. 즉

1. 여러개의 처리기들이 있다.
2. 이 처리기들은 같은 주기억기와 입출력기능들을 공유하며 이것들은 통신모선이나 다른 내부련결방식으로 호상 련결된다.
3. 모든 처리기들이 동일한 기능을 수행할수 있다(이로부터 대칭형이라는 용어를 붙였다.).

SMP의 조작체계는 모든 처리기들에 한하여 프로쎄스들이나 스레드들의 일정작성을 한다. SMP는 단일처리기구성방식에 비하여 다음과 같은것들을 포함하는 몇가지 주요한 우월성을 가지고 있다. 즉

- **성능 :** 콤퓨터가 수행해야 할 작업의 일부를 병렬로 수행할수 있게 조직할수 있다면 다중처리기를 가진 체계가 같은 형의 단일처리기를 가진 체계보다 더 큰

성능을 발휘하게 된다. 이것을 그림 2-12에서 설명하고 있다. 다중프로그람처리에서는 한번에 한개의 프로쎄스만을 집행할수 있다. 그동안 다른 모든 프로쎄스들은 처리기를 기다리고 있다. 다중처리에서는 한개이상의 프로쎄스를 각각 서로 다른 처리기상에서 동시에 실행할수 있다.

- **사용가능성 :** 대칭형다중처리기에서는 모든 처리기들이 같은 기능을 수행할수 있기때문에 한개의 처리기고장이 기계를 정지시키지 않는다. 그대신 체계는 성능이 떨어 진 상태에서 기능을 계속 수행할수 있다.

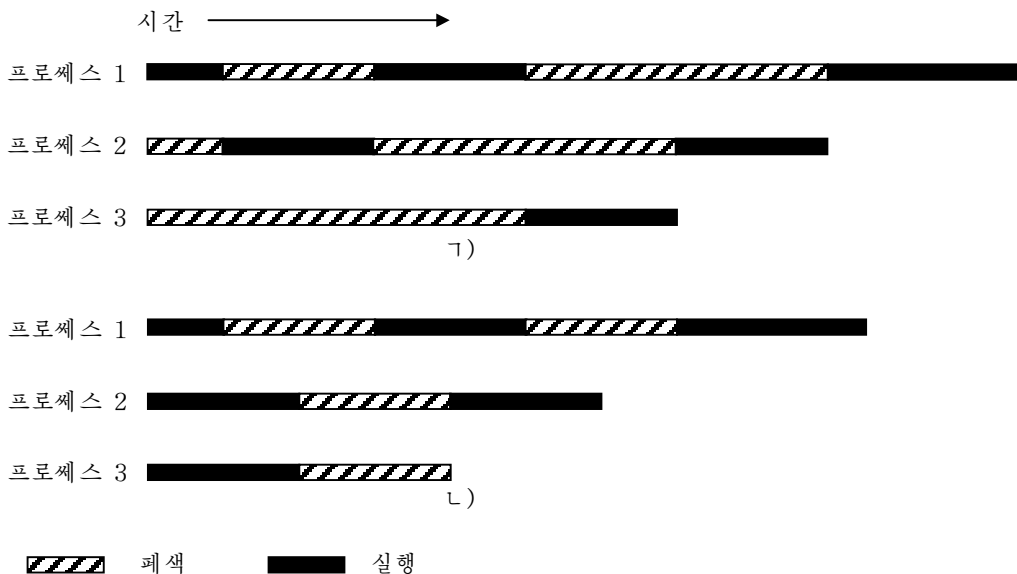



그림 2-12. 다중프로그람처리기와 다중처리 : ㄱ-교차처리(다중프로그람처리;한개의 처리기), ㄴ-교차처리 및 겹침처리(다중처리;다중처리기)

- **증분성장 :** 보충적인 처리기를 추가하여 사용자가 체계의 성능을 강화시킬수 있다.
- **척도맞추기 :** 체계에 배치된 여러개의 처리기들에 기초하여 판매자들이 각이한 가격과 성능특성을 가진 어떤 범위의 제품을 줄수 있다.

중요한것은 이것들이 리익을 담보한다기보다 오히려 잠재적이라는것이다. 조작체계는 SMP체계에서 병렬화를 개발하기 위한 도구와 기능들을 보장해야 한다.

다중스레드처리와 SMP를 흔히 함께 론하면 둘은 독립적인 기능들이다. 단일처리기로 된 기계에서도 다중스레드처리는 응용프로그람과 핵심부처리를 구조화하는데서 쓸모가 있다. SMP기계는 비스레드식프로쎄스들에서 쓸모가 있다. 그것은 몇개의 프로쎄스들을 병렬로 실행할수 있기때문이다. 그러나 두 기능들은 서로 보상되여 효과적으로 함께 사용할수 있다.

SMP의 인기 있는 특징은 여러개 처리기들의 존재가 사용자에게 투명적이라는것이다. 조작체계는 개별적인 처리기들상에서의 스레드나 프로쎄스들의 일정작성이나, 처리기들사이에서의 동기화에 대하여 주의를 돌린다. 이 책에서는 사용자에게 단일체계를 보장하는데 쓰이는 일정작성과 동기화수법을 론의한다. 다른 문제는 개별적인 콤퓨터들의 클라스터 즉 다중콤퓨터체계를 위한 단일체계를 제공하는것이다. 이 경우에는 구체례(콤퓨터들)들의 집합을 취급하는데 매개는 자기자체의 주기억기, 2차기억기와 입출력모듈을

가지고 있다. **분산조작체계**는 단일주기억공간과 단일2차기억공간 및 분산파일체계와 같은 다른 단일화된 접근기능들에 대한 착각을 일으킨다. 비록 클라스터들이 많이 대중화되여 가고 있고 시장에 많은 클라스터제품들이 있다 해도 분산조작체계의 기교상태는 단일처리기와 SMP조작체계의 기교에 뒤떨어 지고 있다. 그러한 체계들은 제6편에서 론의한다.

조작체계설계에서 가장 최근에 발명된것은 객체지향수법을 사용하는것이다. **객체지향설계**는 소형의 핵심부에 모듈적확장을 추가하는 프로쎄스들의 규칙을 보태 준다. 조작체계준위에서 객체에 기초한 구조는 프로그람작성자들이 체계의 완정성을 분해하지 않고 조작체계를 주문할수 있게 한다. 객체지향구조는 또한 분산형도구들과 완성된 분산조작체계의 개발을 쉽게 해준다.

제 5 절. Windows 2000의 개괄

이 절에서는 Windows 2000을 개괄하고 다음절에서 UNIX에 대하여 설명한다. Windows 2000을 간단히 W2K라고 한다.

개발과정

W2K에 대한 이야기는 IBM회사의 첫 개인용콤퓨터용으로 마이크로쏘프트회사가 개발하였는데 MS-DOS 또는 PC-DOS라고 하는 매우 많은 조작체계들로부터 시작한다. 1981년 8월에 초기판본 DOS 1.0이 나왔다. 그것은 4000행의 아셈블리언어의 원천코드로 구성되였으며 Intel 8086극소형처리기를 사용하여 8kbyte의 기억기에서 실행하였다.

IBM회사가 하드디스크에 기초한 개인용콤퓨터 PC XT를 개발했을 때 마이크로쏘프트회사는 DOS 2.0을 개발하여 1983년에 내놓았다. 그것은 하드디스크를 지원하였고 계층구조적인 등록부를 제공하였다. 종래에는 디스크가 최대 64개의 파일을 지원하면서 한개 등록부의 파일들만을 포함할수 있었다. 이것이 유연성디스크시대에서는 충분했지만 하드디스크시대에는 너무 제한되였고 단일한 등록부제약조건은 지내 단순한것이였다. 새로운 판본은 등록부들이 파일들은 물론 보조등록부를 포함하게 하였다. 새로운 판본은 또한 조작체계에 매몰된 보다 풍부한 지령들을 포함시켰는데 이것은 편의프로그람들로 제공된 외부프로그람들에 의하여 수행해야 하였던 기능을 보장하기 위한것이다. 추가된 기능들중에는 주어 진 응용프로그람에서의 입력이나 출력일치성을 변화시킬수 있는 능력인 입출력의 방향바꾸기와 배경인쇄와 같은 몇가지 UNIX와 류사한 기능들이 있었다. 기억기상주부분은 24Kbyte로 증가되였다.

IBM이 1984년에 PC AT를 공포했을 때 마이크로쏘프트회사는 DOS 3.0을 도입하였다. AT는 Intel 80286처리기를 포함하였는데 이것은 확장된 주소지정과 기억기보호기능들을 제공하였다. DOS는 이것을 사용하지 않았다. 이전 제안들과의 호환성을 유지하기 위해 조작체계는 80286을 단순히《고속 8086》과 같이 사용하였다. 조작체계는 새로운 건반과 하드디스크주변장치들을 실질적으로 지원하였다. 기억기요구는 36Kbyte로 늘어 났다. 3.0판에 대한 몇가지 주목할만한 갱신이 있었다. 1984년에 나온 DOS 3.1은 PC들을 망화하는데서의 지원을 포함하였다. 상주부분의 크기는 변화시키지 않았는데 이것은 교체하여 넣을수 있는 조작체계의 량을 증가시켜 얻어 진것이였다. 1987년에 나온 DOS 3.3은 새로운 방법을 가진 IBM기계들, PC/2을 위한 지원을 제공하였다. 그러나 이 판본은 80286과 32bit 80386소편들이 제공하는 PS/2의 처리기능력상 우점을 살리지 못하였다.

이 단계에서 상주부분은 최소 64Kbyte까지 증가했고 부가적인 확장을 선택하면 더 요구되였다.

이 시기까지 DOS는 자기 능력을 훨씬 벗어 난 환경에서 사용되고 있었다. 80486과 Intel Pentium소편의 도입은 단순한 DOS를 가지고 간단히 개발할수 없는 능력과 기능들을 제공하였다. 그동안 1980년대 초엽에 Microsoft회사는 사용자와 DOS사이에 놓이게 될 도형사용자대면부(GUI)의 개발을 시작하였다. Microsoft회사의 목적은 마킨토시회사와 경쟁하는것이였다. 그것은 자기의 조작체계의 사용상 편리성에서 Macintosh조작체계를 릉가하지 못하고 있었기때문이다. 1990년까지 Macintosh회사는 Windows 3.0으로 알려 져 있는 GUI판본을 가지고 있었는데 이것은 Macintosh사용자친절성에 접근한것이였다. 그러나 아직 DOS상에서 실행할데 대한 요구에는 매여 있었다.

새로운 극소형처리기들의 능력을 개발하며 Windows의 사용상 편리한 기능을 병합시킬 다음세대 조작체계[2]를 IBM회사와 함께 개발하려던 Microsoft회사의 시도가 실패한 후에 Microsoft회사는 자체로 그 기초우에서 새로운 조작체계 windows NT를 개발하였다. Windows NT는 현대적인 극소형처리기의 능력을 개척한것으로서 단일사용자 또는 다중사용자환경에서 다중과제처리를 제공하고 있다.

Windows NT의 첫 판본(3.1)은 1993년에 나왔는데 또다른 조작체계(Windows 3.0 계승)인 GUI를 가지고 있다. 그러나 NT 3.1은 새로운 32bit조작체계로서 구식 DOS나 Windows응용프로그람을 지원할수 있는 능력을 가지고 있는것은 물론 OS/2에 대한 지원도 한다.

NT 3.x의 몇개 판본을 내 놓은후에 Microsoft는 NT 4.0을 내 놓았다. NT 4.0은 본질상 3.x와 같은 내부구성방식을 가지고 있다. 가장 주목할만한 외적변화는 NT 4.0이 Windows 98과 같은 사용자대면부를 제공한것이다. 주되는 구성방식상 변화는 3.x에서 win 32부분체계의 일부로서 사용자방식에서 실행했던 몇가지 도형처리성분들을 핵심부방식에서 실행하는 Windows NT의 집행부에로 넘겨 놓은것이다. 이 변화로부터 얻은 리익은 이 중요한 기능들에 대한 조작에서 속도를 높인다는데 있다. 잠재적인 결함은 도형기능들이 이제는 조작체계의 믿음성에 영향을 줄수 있는 낮은 준위체계봉사에 접근한다는것이다.

2000년에 Microsoft회사는 그다음의 주되는 갱신을 도입하였다. 현재 그것을 Windows 2000이라고 부른다. 바탕에 놓여 있는 집행부와 마이크로핵심부의 구성방식은 기본상 NT 4.0에서와 같지만 새로운 기능들이 추가되였다. W2K에서 강조할것은 분산처리를 지원하는 봉사들과 기능들의 추가이다. W2K의 새로운 기능들중에서 중심적요소는 Active directory인데 이것은 객체들에 대하여 임의의 종류의 정보에 임의의 객체들의 이름을 사영할수 있는 분산등록부봉사이다.

W2K에 대한 리해에서 한가지 최종적이며 일반적인 관점은 W2K Server와 W2K Professeional을 구별하는것이다. 본질상 마이크로핵심부와 집행부구성방식 및 봉사들은 같지만 봉사기는 망봉사기로 사용하는데 필요한 일부 봉사들을 포함한다.

단일사용자다중과제처리

W2K는 극소형콤퓨터조작체계에서 새로운 류형에 대한 좋은 실례이다(다른 실례들은 OS/2과 Mac OS이다.). W2K는 바로 몇년전의 대형콤퓨터나 소형콤퓨터들과 속도, 하드웨어구성 및 기억용량상에서 그것들과 맞먹는 오늘날의 32bit극소형처리기의 처리능

---

2. IBM회사는 자체로 OS/2을 계속 개발하였다. Windows NT와 같이 OS/2은 다중과제처리, 다중스레드조작체계이다.

력을 개척할데 대한 요구로부터 개발되였다.

이 새로운 조작체계의 가장 중요한 특징중의 하나는 비록 그것이 아직은 단일대화형 사용자를 지원하고 있지만 다중과제처리조작체계라는것이다. 두개의 기본적인 개발인 개인용콤퓨터, 워크스테이션과 봉사기상에서 다중과제처리를 할데 대한 요구를 촉발시켰다. 우선 극소형처리기의 속도와 기억기용량이 증가함에 따라 가상기억기, 응용프로그람에 대한 지원은 보다 복잡하고 호상 련관되게 되였다. 실례로 사용자가 문서를 만들기 위하여 단어처리기, 작도프로그람, 자료표응용프로그람을 동시에 사용하려고 할수 있다. 다중과제처리가 없이 사용자가 작도를 하고 그것을 문서처리하는 본문에 붙이려고 한다면 다음과 같은 단계가 요구된다. 즉

1. 작도프로그람의 열기
2. 작도하고 그것을 파일이나 림시 소개지에 보관하기
3. 작도프로그람의 닫기
4. 문서처리프로그람의 열기
5. 작도한것을 정확한 위치에 삽입하기

수정할 필요가 있다면 사용자가 문서처리프로그람은 닫고 작도프로그람을 열고 도형화상을 편집하며 그것을 보관하고 작도프로그람을 닫고 문서처리프로그람을 열고 수정된 화상을 삽입하여야 한다. 이것은 인차 지루감을 주게 된다. 사용자에게 줄수 있는 봉사와 능력이 보다 강력해 지고 다양해 짐에 따라 단일과제환경은 보다 멋 없고 사용자에게 친근감을 주지 못한다. 다중과제처리환경에서는 사용자가 매개 응용프로그람을 필요에 따라 열고 열려있는채로 그것을 탈퇴한다. 정보는 여러 응용프로그람들사이에서 쉽게 이동할수 있다. 매개 응용프로그람은 한개이상의 열린 창문을 가지고 있으며 마우스와 같은 지시장치를 가진 도형대면부가 사용자로 하여금 이 환경에서 쉽게 항행할수 있게 해준다.

다중과제처리의 두번째 필요성은 의뢰기/봉사기계산이 증가한데 있다. 의뢰기/봉사기계산에서 개인용콤퓨터나 워크스테이션(의뢰기)과 주체계(봉사기)는 어떤 특정한 응용프로그람을 위해 련합하여 쓰인다. 그것들은 서로 련결되며 각자는 자기능력에 맞는 일감묶을 할당 받는다. 의뢰기/봉사기는 개인용콤퓨터의 국부지역망과 봉사기 또는 사용자체계가 대형콤퓨터와 같은 대형주체계에서 목적을 달성할수 있다. 어떤 응용프로그람은 한개이상의 개인용콤퓨터와 한개이상의 봉사기를 포함할수 있다. 요구되는 응답성을 보장하기 위하여 조작체계는 조화로운 실시간통신하드웨어와 그리고 관련된 통신규약 및 같은 시간에 사용자와 대화를 계속 유지할수 있게 하는 자료이송방식을 제공해야 한다.

앞에서 고찰한 사항들은 W2K의 전문판본에 적용한다. 봉사기판본역시 다중과제처리이지만 다중사용자들을 지원할수 있다. 그것은 다중말단봉사기련결은 물론 망상의 여러 사용자들이 사용하게 되는 공유자원제공을 지원한다. 인터네트봉사기에서와 같이 W2K는 수천개의 동시적인 Web련결을 지원할수 있다.

구성방식

그림 2-13은 [SOLD98b]의 내용에 기초하여 W2K의 총적인 구조를 설명하고 있다. 그것의 모듈식구조는 W2K에 상당한 유연성을 준다. 그것은 여러가지의 하드웨어가동환경상에서 집행하도록 설계되여 있으며 여러가지 다른 조작체계용으로 작성된 응용프로그람들을 지원한다. 이와 같이 작성하면 W2K는 Pentium/X86하드웨어가동환경상에서만 실현된다.

가상적으로 모든 조작체계들과 같이 W2K는 조작체계쏘프트웨어로부터 응용프로그람지향쏘프트웨어를 분리시킨다. 집행부, 마이크로핵심부, 장치구동프로그람구체례, 하드

웨어추상화층을 포함하는 후자는 핵심부방식으로 실행한다. 핵심부방식쏘프트웨어는 체계자료와 하드웨어에 접근한다. 사용자방식에서 실행하는 나머지쏘프트웨어는 체계자료에 대한 접근에서 제한을 가진다.

조작체계의 조직

W2K는 순수한 마이크로핵심부구성방식이 아니라 마이크로쏘프트가 변경한 마이크로핵심부구성방식을 가지고 있다. 순수한 마이크로핵심부구성방식에서와 같이 W2K은 모듈화가 잘 되여 있다. 매개 체계기능은 조작체계의 바로 한개 요소가 관리한다. 조작체계의 나머지부분과 모든 응용프로그람들은 표준대면부를 사용하여 응답할수 있는 요소들을 거쳐 그 기능에 접근한다. 적당한 기능을 통해서만 체계자료에 접근할수 있다. 원리적으로 전체체계나 그의 표준응용프로그람대면부(API)를 다시 작성하지 않고 임의의 모듈을 이동, 갱신 또는 교체할수 있다. 그러나 순수한 마이크로핵심부체계와 달리 W2K는 마이크로핵심부바깥쪽에 있는 많은 비마이크로핵심부기능을 사용하는것이 몇개의 프로쎄스 또는 스레드의 절환, 방식절환 및 추가적인 기억완충기를 사용할것을 요구한다는것을 발견하였다.

W2K설계목표의 하나는 이식성인데 그것이 꼭 인텔기계상에서가 아니라 임의의 하드웨어가동환경상에서 실행할수 있게 한다. 이 목표를 달성하기 위하여 대부분의 W2K집행부는 다음과 같은 층구조를 사용하여 바탕에 놓여 있는 하드웨어를 같은 관점에서 보고 있다. 즉

- **하드웨어추상층(HAL) :** 일반하드웨어지령과 응답 및 특정한 가동환경에 유일한 것들사이에서 사영한다. 그것은 특정한 가동환경의 하드웨어의 차이로부터 조작체계를 고립시킨다. HAL은 매개 기계의 체계모선, 직접기억접근(DMA)조종기, 새치기조종기, 체계시계와 기억기모듈을 핵심부에 대해 같은것으로 보게 한다. 그것은 또한 다음에 설명하는 대칭다중처리(SMP)에 필요한 지원을 준다.
- **마이크로핵심부 :** 조작체계중에서 가장 많이 쓰이며 기본적인 성분들로 구성되여 있다. 핵심부는 스레드일정작성, 프로쎄스절환, 례외 및 새치기조종, 다중극소형처리기의 동기화를 관리한다. 집행부와 사용자준위의 나머지부분과 달리 마이크로핵심부자체의 코드는 스레드에서 실행하지 않는다. 이로부터 그것은 조작체계의 한 부분일따름이고 선취하거나 페지화할수 없다.
- **장치구동프로그람 :** 파일체계와 그리고 사용자입출력기능호출을 특정한 하드웨어장치입출력요청으로 번역하는 하드웨어장치구동프로그람을 포함한다.

W2K집행부는 특정한 체계기능을 위한 모듈을 포함하며 사용자방식쏘프트웨어를 위한 API를 제공한다. 다음의것들이 매개 집행부모듈에 대한 요점적인 설명으로 된다. 즉

- **입출력관리자 :** 입출력장치나 응용프로그람에 접근할수 있는 구조를 주며 앞으로의 알맞는 장치구동프로그람구체례를 배분할 책임을 진다. 입출력관리자는 모든 W2K의 입출력 API를 실현하고 보안을 실시하며 장치와 파일체계들에 이름을 달아 준다(객체관리자를 사용하여). W2K입출력에 대해서는 제11장에서 설명한다.
- **객체관리자 :** 프로쎄스, 스레드 및 동기화객체와 같은 자원들을 표현하는데 쓰이는 W2K집행부준위객체들과 추상적인 자료형들을 창조, 관리, 삭제한다. 그것은 객체들에 대한 보안을 유지하고 이름을 붙이며 설정하기 위한 통일적인 규칙들을 시행한다. 객체관리자는 또한 객체조종기를 창조하는데 이것은 접근조종정보와 객체에 대한 지시기로 구성되여 있다. W2K의 객체들은 이 절에서 후에 설명한다.

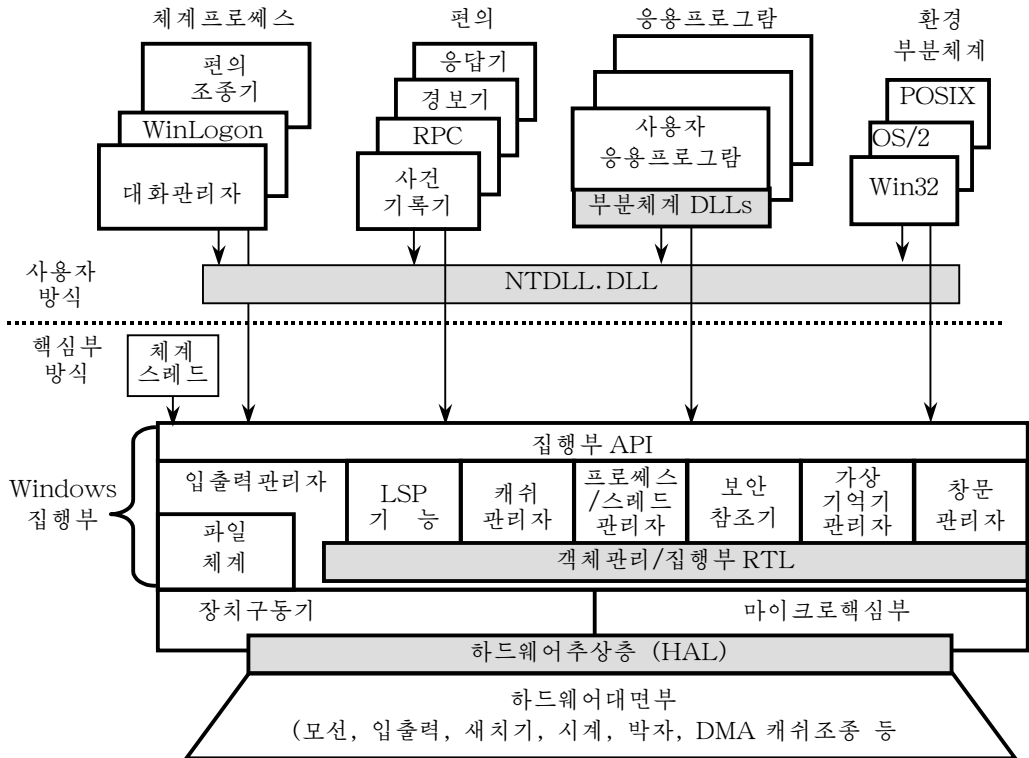



그림 2-13. Windows 2000의 구성방식

- **보안참조감시기** : 접근유효화와 검열발생규칙을 시행한다. W2K의 객체지향모형은 보안에 대한 일관적이면서 유일적인 관점에서 기본구체레들에 의하여 집행부를 만들도록 한다. 이렇게 함으로써 W2K는 접근유효화를 위하여 그리고 파일, 프로쎄스, 주소공간 및 입출력장치를 포함하는 모든 보호객체들을 위한 시청검열을 위하여 같은 루틴을 사용한다. W2K의 보안에 대해서는 제15장에서 설명한다.
- **프로쎄스/스레드관리자** : 객체들을 생성하고 삭제하면 프로쎄스 및 스레드객체들을 추적한다. W2K의 프로쎄스와 스레드관리에 대해서는 제4장에서 설명한다.
- **국부수속호출(LPC)기능** : 단일체계에 있는 응용프로그람과 집행부분체계사이의 의뢰기/봉사기관계를 분산처리에 사용하는 원격수속호출(RPC)기능과 류사한 방식으로 시행한다.
- **가상기억기관리자** : 처리기주소공간의 가상주소를 콤퓨터기억기의 물리적페지에로 주소사영한다. W2K의 가상기억기관리에 대해서는 제8장에서 서술한다.
- **캐쉬관리자** : 최근에 참조한 디스크자료를 고속접근용주기억기에 상주시키게 하고 그리고 디스크에 보내기전에 짧은 시간동안 기억기에서의 갱신을 진행하여 디스크쓰기를 지연시키게 하는 방법으로 파일에 기초한 입출력의 성능을 개선한다.
- **창문/도형모듈** : 창문지향화면대면부를 생성하며 도형장치를 관리한다.

사용자프로쎄스

W2K는 네가지 기본형태의 사용자프로쎄스를 지원한다. 즉

- **특수체계지원프로쎄스 :** 가입프로쎄스 및 대화관리자와 같은 W2K조작체계의 부분으로서 제공되지 않은 봉사를 포함한다.
  - **봉사기프로쎄스 :** 사건기록기와 같은 다른 W2K봉사를 제공한다.
- **환경부분체계 :** 본래의 W2K봉사를 사용자응용프로그람에 로출시켜 조작체계환경 또는 개인성을 제공한다. 지원하고 있는 부분체계는 Win 32, Posix와 OS/2이다. 매개 환경부분체계는 동적련결서고(DLL)를 포함하는데 이것은 사용자프로그람호출을 W2K호출로 변환한다.
- **사용자응용프로그람 :** Win 32, Posix, OS/2, Windows 3.1 또는 MS-DOS 등의 다섯가지 형태들중 한가지가 될수 있다.

W2K는 구조화되여 있어 W2K, Windows 98 및 몇가지 다른 조작체계용으로 작성된 응용프로그람들을 지원한다. W2K는 보호부분체계를 통하여 한개의 밀집된 집행부를 사용함으로써 그러한 지원을 준다. 보호된 부분체계들은 W2K의 부분으로서 일반사용자와 대화한다. 매개 부분체계는 프로쎄스를 분리시키며 집행부는 그의 주소공간을 다른 부분체계와 응용프로그람의 주소공간으로부터 보호한다. 보호된 부분체계는 도형 또는 지령행의 사용자대면부를 제공하는데 이것은 사용자에 대한 조작체계의 보기와 느낌을 정의한다. 또한 보호된 부분체계는 특정한 조작환경을 위한 API를 준다. 이것은 특정한 조작환경을 위하여 창조한 응용프로그람을 W2K상에서 변화시키지 않고 실행시킬수 있다는것을 의미한다. 왜냐하면 그들이 보는 조작체계대면부가 그들이 작성한것과 같기때문이다. 그래서 실례를 들면 OS/2에 기초한 응용프로그람은 변경하지 않고 W2K조작체계하에서 실행할수 있다. 더우기 W2K체계는 하드웨어추상층(HAL)을 사용하여 독립적인 가동환경이 되도록 그자체가 설계되여 있으므로 보호된 부분체계와 그것들이 지원하는 응용프로그람들을 모두 한개의 하드웨어가동환경으로부터 다른 하드웨어가동환경에로 옮기기가 상대적으로 쉬워 진다. 많은 경우에 재콤파일이 항상 제기된다.

가장 중요한 부분체계는 Win 32이다. Win 32는 W2K와 Windows 98량자우에서 실현된 API이다. Win 32의 일부 기능들은 Windows 98에서 실현할수 없지만 Windows 98상에서 실현된 기능들은 W2K의것과 동일하다.

의뢰기/봉사기모형

집행부, 보호된 부분체계와 응용프로그람들은 의뢰기/봉사기계산모형을 사용하여 구조화되는데 이 모형은 분산계산의 일반적모형으로서 제6편에서 설명한다. 이 구성방식을 단일체계내부에서 사용하도록 받아 들일수 있다. W2K의 경우가 바로 그렇다.

매개 환경부분체계와 집행부봉사부분체계는 한개 또는 그이상의 프로쎄스들로 실현된다. 매개 프로쎄스는 그것의 봉사들중(실례로 기억기봉사, 프로쎄스창조봉사 또는 처리기일정작성봉사) 하나에 대한 의뢰기의 요청를 기다린다. 응용프로그람이나 또는 다른 조작체계모듈일수 있는 의뢰기는 통보문을 보내는 방법으로 어떤 봉사를 요청한다. 그 통지문은 집행부를 통하여 해당한 봉사기에로 들어 간다. 봉사기는 요청되는 동작을 수행하고 결과나 상태정보를 다른 통보문형식으로 되돌려 보낸다. 이 통보문은 집행부를 통하여 의뢰기에로 되돌아 간다.

의뢰기/봉사기구성방식의 우점에는 다음과 같은것들이 있다. 즉

표 2-5. Win 32 API가 포괄하는 몇가지 령역[RICH97]

원자	망
자식조종	흐름관과 우편홈
오림판조작	인쇄
통신	프로쎄스와 스레드
조종탁	등록자료기지조작
오유수정	자원
동적련결서고(DLLs)	보안
사건기록	봉사
파일	구조화된 례외조종
도형작도기본지령	체계정보
건반 및 마우스입력	테프여벌
기억기관리	시간
다매체봉사	창문관리

- 집행부를 간소화한다. 집행부에서 충돌이나 중복이 없이 여러가지 APL를 구성할 수 있다. 또한 새로운 APL을 쉽게 추가할수 있다.
- 믿음성을 개선한다. 매개 집행부봉사모듈은 개별적인 프로쎄스상에서 실행되며 그것이 차지한 기억기의 부분은 다른 모듈로부터 보호된다. 하드웨어에 접근하거나 집행부가 보관되여 있는 기억기를 변경할수 없다. 단일의뢰기는 조작체계의 나머지부분을 파산시키거나 변경시킴이 없이 실패할수 있다
- 응용프로그람은 유연성의 제한을 받지 않고 LPC를 거쳐 집행부와 통신할수 있는 유일적인 수단을 제공한다. 통보문을 넘겨 주는 프로쎄스는 기능그루터기들에 의하여 의뢰기응용프로그람으로부터 교갑화되여 있는데 이 그루터기들은 동적련결서고(DLL)에 있으면서 집행할수 없는 위치유지기들이다. 응용프로그람이 환경부분체계를 APL호출할 때 의뢰기응용프로그람의 그루터기는 호출을 위한 파라메터들을 묶어 통보문으로 그것들을 봉사기부분체계에 보내여 호출을 실현한다.
- 분산계산을 위한 합리적인 기준을 준다. 대체로 분산계산은 의뢰기/봉사기모형을 사용하는데 원격수속호출은 분산되여 있는 의뢰기와 봉사기모듈과 의뢰기와 봉사기사이에서 통보문교환을 사용하여 실현된다. W2K에서 국부봉사기가 국부의뢰기응용프로그람을 대신하여 처리하기 위하여 통보문을 원격봉사기에 넘겨 줄수 있다. 의뢰기들은 어떤 요청이 국부적으로 봉사를 받았는가 아니면 원격으로 봉사를 받았는가 하는것을 알수 없다. 실제로 어떤 요청이 국부적으로 봉사 받는가 아니면 원격으로 봉사 받는가 하는것은 현재의 적재조건과 동적구성변화에 기초하여 동적으로 변할수 있다.

스레드와 SMP

W2K의 두가지 중요한 기능은 스레드와 대칭다중처리(SMP)에 대한 지원인데 그것들은 이미 제2장 제4절에서 소개하였다. [CUST93]에서는 스레드와 SMP를 지원하는 W2K의 다음과 같은 기능들을 목록화하고 있다.

- 조작체계루틴들은 사용할수 있는 임의의 처리기상에서 실행할수 있으며 각이한

루틴들은 서로 다른 처리기들상에서 동시에 집행할수 있다.
- W2K는 한개의 프로쎄스에서 집행중의 다중스레드를 사용할수 있게 한다. 같은 프로쎄스의 다중스레드는 각이한 처리기상에서 동시에 집행할수 있다.
- 봉사기프로쎄스들은 다중스레드를 사용하여 한개이상의 의뢰기에서 오는 요청을 동시에 처리할수 있다.
- W2K는 프로쎄스들과 유연한 프로쎄스간 통신능력사이에서 자료와 자원을 공유하기 위한 수단을 준다.

Windows 2000의 객체

W2K는 객체지향설계에 대한 개념을 많이 내포하고 있다. 이 방법은 프로쎄스들사이에서 자원과 자료의 공유와 권한이 부여되지 않는 접근으로부터 자원보호를 촉진시킨다. W2K가 사용한 객체지향개념들속에는 다음과 같은것들이 있다. 즉

- **밀봉 :** 객체는 속성이라고 하는 한개 또는 그이상의 자료항목과 그 자료에 기초하여 수행할수 있는 한개 또는 그이상의 수속(봉사라고 한다.)으로 구성되여 있다. 오직 객체내의 자료에 접근하는 방법은 객체의 봉사들중의 하나를 기동하는데 있다. 이와 같이 하여 권한이 부여되지 않은 사용과 부정확한 사용(례를 들면 집행할수 없는 일부 자료를 집행하려는 시도)으로부터 객체의 자료를 쉽게 보호할수 있다.
- **객체클라스와 구체례 :** 객체클라스는 객체의 속성과 봉사를 목록화하며 일정한 객체특성을 정의하는 본보기이다. 조작체계는 요구에 따라 객체클라스의 특정한 구체례들을 창조할수 있다. 실례로 단일프로쎄스객체클라스와 현재 능동인 매개 프로쎄스를 위한 한개의 프로쎄스객체가 있다.이 방법은 객체창조와 관리를 간단하게 한다.
- **계승 :** 이것이 사용자준위에서 지원되지는 않지만 집행부에서 일정한 범위를 지원한다. 실례로 등록부객체는 포함기객체들에 대한 실례이다. 포함기객체의 한가지 속성은 그것들이 포함하는 객체들이 포함기 그자체로부터 속성들을 계승할수 있다는것이다. 실례로 자기의 압축된 기발모임을 가지는 파일체계에 어떤 등록부를 가지고 있다고 가정하자. 그러면 그 등록부포함기에서 창조할수 있는 임의의 파일들은 역시 압축된 기발모임을 가지게 된다.
- **다형성 :** 내적으로 W2K는 어떤 공통적인 모임으로 되여 있는 API기능들을 사용하여 임의의 형태의 객체들을 조작한다. 이것이 바로 부록 2에서 정의하고 있는바와 같이 다형성의 특징이다. 그러나 W2K는 특정한 객체형에 대해 많은 API들이 있기때문에 완전히 다형적인것은 아니다.

객체지향개념에 익숙하지 못한 독자는 이 책의 마감에 있는 부록 2를 자세히 보면 된다.

W2K에서 구체례들모두가 객체인것은 아니다. 사용자방식접근에서 자료를 여는 경우에 또는 자료접근을 공유하거나 제한할 때 객체들을 사용한다. 객체들이 표현하는 구체례들중에는 파일, 프로쎄스, 스레드, 신호기, 시계와 창문들이 있다. W2K는 객체관리자를 거쳐 모든 형태의 객체들을 통일적으로 창조하고 관리한다. 객체관리자는 응용프로그람을 대신하여 객체들을 창조하고 대비하며 객체의 봉사와 자료에 대한 접근을 허용한다.

집행부의 매개 객체를 때로는 핵심부객체(집행부와 관련이 없는 사용자준위객체들과 구별하기 위하여)라고 하며 그 매개 객체는 핵심부가 배정한 기억기와 같이 존재

하며 핵심부만이 접근할수 있다. 자료구조의 일부 요소(실례로 객체이름,보안파라메터, 사용회수)들은 모든 객체형태에 공통이지만 다른 요소들은 객체형태(실례로 스레드객체의 우선권)에 한정된다. 이 핵심부객체자료구조는 핵심부만이 접근할수 있다. 즉 응용프로그람이 자료구조를 찾거나 그것들을 직접 읽거나 쓰거나 할수 없다. 그대신 응용프로그람은 집행부가 지원하는 객체조작기능의 모임을 통하여 간접적으로 객체들을 조작한다. 어떤 객체를 창조할 때 창조를 요청하는 응용프로그람은 객체를 위한 조종기를 도로 받는다. 본질상 조종기는 참조되는 객체에 대한 지시자이다. 프로쎄스의 임의의 스레드가 이 조종기를 사용하여 객체들과 작업하는 Win 32의 기능들을 기동시킬수 있다.

객체들은 그것들과 관련되는 보안정보를 보안분류항목(SD)형태로 가질수 있다. 이 보안정보를 사용하여 객체에 대한 접근을 제한할수 있다. 실례로 어떤 프로쎄스가 이름 붙은 신호기객체를 창조할수 있는데 그 목적은 일정한 사용자들만이 신호기를 열고 사용할수 있게 하자는것이다. 신호기객체용 SD는 허용된 접근의 종류(읽기, 쓰기, 변경 등)에 따라 신호기객체에 대한 접근을 허용 받은(또는 거부된) 사용자들을 목록화할수 있다.

W2K에서 객체들은 이름을 붙일수도 있고 붙이지 않을수도 있다. 프로쎄스가 이름붙지 않은 객체를 창조할 때 객체관리자는 그 객체에로 조종기를 복귀시키며 이때 조종기는 그것을 참조하기 위한 유일한 수단으로 된다. 이름 붙은 객체들은 다른 프로쎄스들을 사용하여 그 객체에 대한 조종기를 획득하는데 사용할수 있는 이름을 가진다. 실례로 프로쎄스 A가 프로쎄스 B와 동기를 맞추려고 한다면 이름 붙은 사건객체를 창조할수 있고 그 사건의 이름을 B에 넘겨 줄수 있다. 그러면 프로쎄스 B는 그 사건객체를 열고 사용한다. 그러나 A가 단순히 사건을 사용하여 자체의 두개의 스레드를 동기화하려 한다면 이름붙지 않은 사건객체를 생성할수 있다. 왜냐하면 다른 프로쎄스들이 그 사건을 사용할수 있게 하는 아무런 요구도 없기때문이다.

W2K가 관리하는 객체의 실례로서 마이크로핵심부가 관리하는 객체들에 대한 두가지 범주를 목록화하고 있다. 즉

표 2-6. NT마이크로핵심부조종객체들[MS96]

비동기	명시된 스레드집행에 갑자기 들어 가거나 어떤 수속을 명시된 처리기방식에서 호출할수 있게 하는데 사용한다.
새치기	새치기원천을 새치기배분표(IDT)에 있는 입구점을 사용하여 새치기봉사루틴에 련결시키는데 사용한다. 매개 처리기는 IDT를 가지고 있는데 이것은 처리기에 발생하는 새치기들을 배분하는데 사용된다.
프로쎄스	한조의 스레드객체의 집행에 필요한 가상주소공간과 조종정보를 표시한다. 프로쎄스는 주소사영에 대한 지시자, 스레드객체들을 포함하는 준비된 스레드의 목록, 프로쎄스에 속하는 스레드목록, 프로쎄스에서 모든 스레드를 집행하기 위한 총적인 축적시간 및 기준우선권을 포함하고 있다.
프로그람실행의 통보	코드블로크에서 실행시간분산을 측정하는데 쓰인다. 사용자와 체계코드는 둘다 실행통보될수 있다.

- **조종객체 :** 배분과 동기화에 영향을 주지 않는 범위에서 마이크로핵심부의 조작을 조종하는데 사용된다. 표 2-6은 마이크로핵심부의 조종객체를 목록화한것이다.
- **배분기객체 :** 체계조작의 배분과 동기화를 조종한다. 제6장에서 이것을 서술한다.

W2K는 완성된 객체지향형조작체계는 아니다. 그것은 객체지향언어로 실현되지 않는다. 한개의 집행부요소에 완전히 상주하는 자료구조는 객체로 표현하지 않는다. 그럼에도 불구하고 W2K는 객체지향기능의 위력을 발휘하고 있으며 조작체계설계에서 이 수법을 사용하려는 시도가 높아 지고 있다.

제 6 절. 전통적인 UNIX체계

개발과정

UNIX의 개발과정에 대해서는 이미 알려 져 있으므로 여기서는 깊은 세부를 반복하지 않는다. 대신 주요줄거리를 강조한다. 중요한것들을 그림 2-14에서 설명하고 있다. 이것은 [SAL94]에 있는 그림에 기초하고 있다.[3]

UNIX는 초기에 Bell연구소에서 개발되여 1970년에 PDP-7상에서 조작하게 되였다.

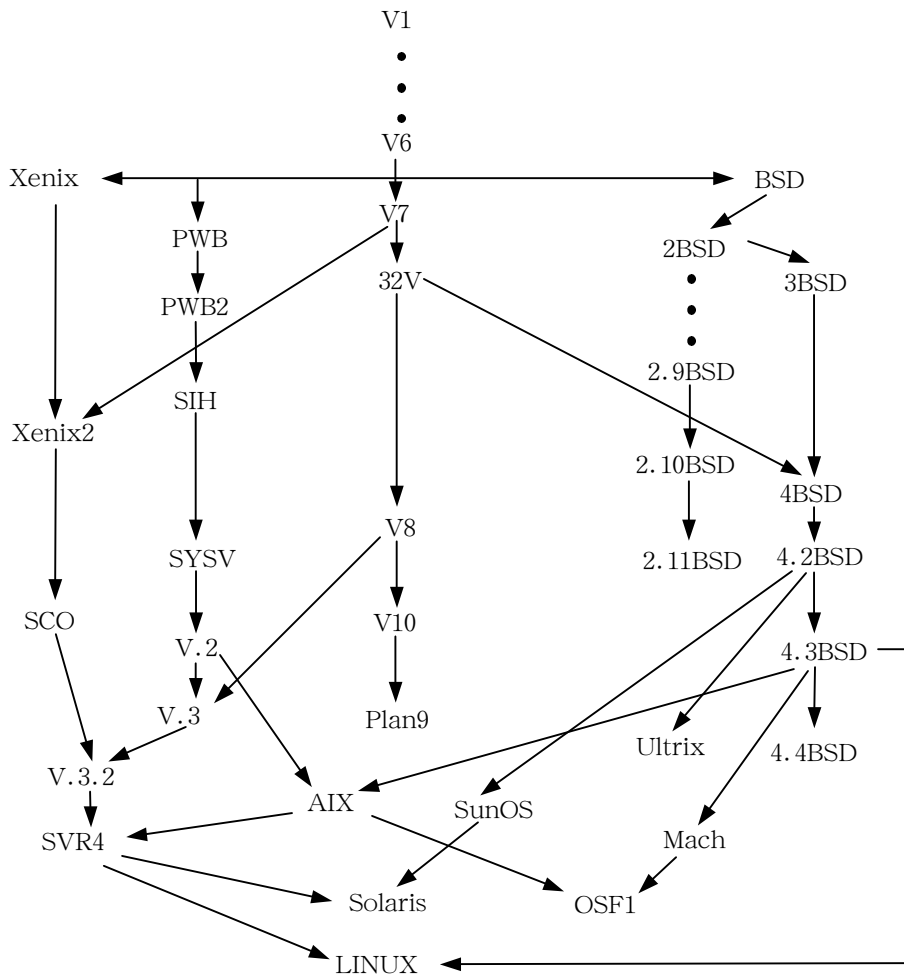



그림 2-14. UNIX 의 개발과정

3. 보다 완전한 계렬나무를 [MCKU96]에서 보여 주고 있다.

Bell연구소에 소속된 일부 사람들은 MIT의 MAC계획에서 진행하는 시분할대상계획에도 참가하였다. 그 계획은 처음 CTSS를 개발하고 다음 Multics를 개발하기로 되여 있었다. 일반적으로 말하기를 UNIX가 Multics를 어떤 비률로 낮춘 판본이라 할지라도 UNIX의 개발자들은 실제적으로 CTSS가 보다 큰 영향을 준다고 주장하였다. 그렇지만 UNIX는 Multics의 구상을 많이 받아 들이였다.

Bell연구소에서 UNIX에 대한 연구와 그후 그밖의 연구사업을 통하여 여러가지 UNIX계렬품들을 내놓았다. 처음으로 주목할만한 획기적전변은 UNIX체계를 PDP-7로부터 PDP-11에로 방향을 돌린것이였다. 이것은 UNIX가 모든 콤퓨터용조작체계로 될것이라는 첫 암시였다. 다음으로 중요한 전변은 프로그람작성언어 C로 UNIX를 다시 작성한 것이였다. 이것은 그 당시에 들어 보지 못한 전략이였다. 일반적으로 그것은 조작체계와 같이 복잡한것으로 느껴 졌다. 왜냐하면 그것이 시간적으로 엄격한 사건들을 취급해야 하며 아쎔블리언어로 배타적으로 작성되여야 하였기때문이다. C의 실현은 체계코드의 전부가 아니라 대부분에 고급언어를 사용하는것이 가지는 우월성을 증명하였다. 오늘날 실제적으로 UNIX실현물들은 모두 C로 작성된다.

UNIX의 초기판본들은 Bell연구소에서 완전히 대중화되였다. 1974년에 UNIX체계는 처음으로 기술잡지에 실렸다[RITC74].이것은 체계에 대한 큰 흥미를 불러 일으켰다. UNIX의 허가증이 광고협회는 물론 종합대학들에까지 제공되였다. Bell연구소가 내 놓은데서 널리 사용할수 있는 첫 판본은 1976년에 나온 판본 6이였다. 계속하여 1978년에 나온 판본 7은 가장 현대적인 UNIX체계의 조상이다. 비 AT&T체계를 개발하는데서 가장 중요한것은 버클리에 있는 캘리포니아종합대학에서 수행되였는데 그것을 UNIX BSD라고 불렀고 처음에는 PDP상에서 실행하고 다음은 VAX기계들에서 실행하였다. AT&T는 계속 체계를 개발하고 세련시켰다. 1982년경에 Bell연구소는 UNIX에 대한 몇가지 AT&T방안들을 단일체계에 포함시켜 AT&T System Ⅲ으로 상업화하였다. UNIX System Ⅴ를 만들기 위해 후에 몇가지 기능들이 조작체계에 추가되였다.

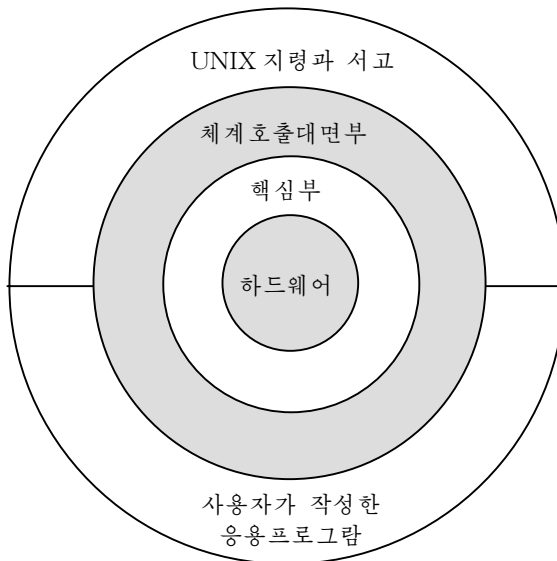



그림 2-15. 일반적인 UNIX 구성방식

해설

그림 2-15는 UNIX구성방식에 대한 일반설명을 보여 주고 있다. 바탕에 놓여 있는 하드웨어는 조작체계쏘프트웨어로 둘러 싸여 있다. 조작체계를 사용자와 응용프로그람으로부터 고립시킨다는것을 강조하기 위해 조작체계를 흔히 체계핵심부 간단히 핵심부라고 한다 UNIX의 이 부분은 이 책에서 UNIX를 실례로 사용하는데서 그 내용과 련관이 있다.그러나 UNIX는 체계의 부분으로 간주되는 많은 사용자봉사와 대면부들로 장비되여 있다. 이것들을 쉘, 다른 대면부쏘프트웨어와 C콤파일러(콤파일러,아쎔블러,적재기)의 구성요소로 묶어 놓을수 있다. 이것의 바깥층은 사용자응용프로그람과 C콤파일러에 대한 사용자대면부로 구성되여 있다.

핵심부를 더 구체적으로 본것이 그림 2-16에 제시되여 있다. 사용자프로그람은 직접적으로 또는 서고프로그람을 통해서 조작체계봉사를 요청할수 있다. 체계호출대면부는 사용자와 경계를 이루며 높은 준위 쏘프트웨어가 명시한 핵심부기능을 호출할수 있게 한다. 다른 한편 조작체계는 하드웨어와 직접 대화하는 원시적인 루틴들을

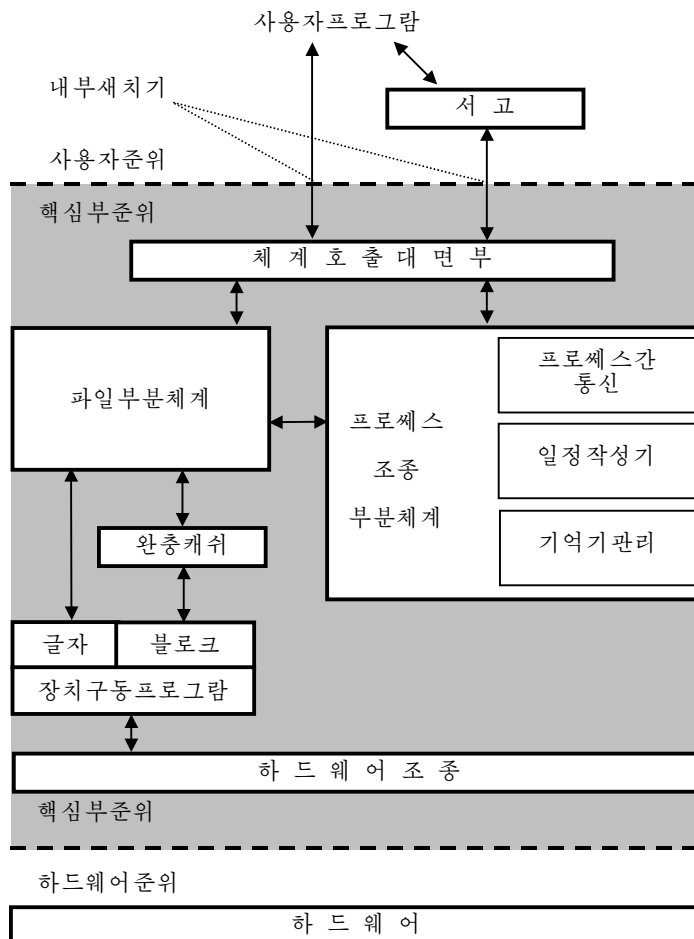



그림 2-16.전통적인 UNIX 핵심부[BACH86]

포함하고 있다. 이 두 대면부사이에서 체계는 두개의 주요부분으로 나누어 지는데 하나는 프로쎄스조종과 관련된것이고 다른 하나는 파일관리 및 입출력관리와 관련된것이다. 프로쎄스조종부분체계는 기억기관리, 프로쎄스들의 일정작성 및 배분, 프로쎄스들의 동기화 및 프로쎄스사이 통신에 대해 책임 진다. 파일체계는 기억기와 외부장치사이에서 문자흐름으로 또는블로크별로 자료를 교환한다. 이를 위해 여러가지 장치구동프로그람구체례를 사용한다. 블로크지향이송을 위해 디스크개시방법을 사용한다. 즉 주기억기내의 체계완충기를 사용자주소공간과 외부장치사이에 끼워 넣는다.

이 부분절에서는 전통적인 UNIX체계라고 말할수 있는것들을 취급한다. [VAHA 96]에서는 전통적이라는 용어를 System V Release 3(SVR3), 4.3BSD 및 초기판본들을 가리키는데 쓰고 있다. 그것은 단일처리기상에서 실행하도록 설계되여 있고 따라서 여러처리기들이 병행하여 호출하면 자료구조를 보호할 능력이 없다. 핵심부는 만능적인것이아니며 단일형태의 파일체계, 프로쎄스일정작성방법, 집행가능한 파일서식을 준다. 고전적인 UNIX핵심부는 확장할수 있게 설계되지 않았으며 코드재사용을 위한 몇가지 기능들만 가지고 있다. 결과 새로운 기능들을 여러가지 UNIX판본들에 추가함에 따라 많은 새로운 코드들을 추가해야 했고 결국은 팽창되여 모듈화되지 않은 핵심부를 낳게 하였다.

제 7 절. 현대적인 UNIX체계

UNIX가 발전함에 따라 몇가지 서로 다른 실현물들이 결말을 보았는데 매개는 일부 유용한 기능들을 주고 있다. 많은 주요한 발명들을 단일화하고 다른 현대적인 OS설계특징들을 추가하며 보다 모듈화된 구성방식을 주고 새로운 실현물을 내 놓을데 대한 요구가 있었다. 대표적인 현대 UNIX핵심부를 그림 2-17에서 묘사하고 있다. 모듈방식으로 작성된 작은 봉사핵심이 있으며 그것은 많은 OS프로쎄스들이 요구하게 될 기능과 봉사들을 준다. 바깥의 원들은 각각 여러가지 방식으로 실현할수 있는 기능들과 대면부를 표시한다.

현대 UNIX체계들에 대한 일부 실례들을 보기로 하자.

System V Release 4(SVR 4)

SVR 4는 AT&T의 Sun Microsystems회사가 련합하여 개발하였는데 이것은 SVR 3, 4.3BSD, Microsoft Xenix systme V 및 SUN OS를 결합시킨다. 그것은 거의 모두가 Systme V핵심부에 대한 총체적인 재작성이였고 복잡하지만 균형이 잡힌 실현물을 내놓았다. 제안에서 새로운 특징은 실시간처리지원, 프르세스일정작성클라스, 동적으로 배정된 자료구조, 가상기억기관리, 가상파일체계와 선취핵심부를 포함하는것이다.

SVR 4는 상업적이며 학구적인 설계가들 량자의 노력에 의해 나왔으며 상업적인 UNIX전개를 위한 통일적인 가동환경을 제공하기 위하여 개발되였다. 그것은 이 객체에서 성공하였으며 아마도 지금까지의 가장 중요한 UNIX방안으로 된다. 그것은 UNIX체계상에서 지금까지 개발된 대부분의 중요한 기능들을 병합시키고 있으며 또 상업적으로 생활력 있는 방식에서도 역시 그렇게 하고 있다. SVR 4는 32bit극소형처리기로부터 초고속콤퓨터에 이르기까지의 모든 기계들에서 실행되고 있으며 지금까지 개발된 가장 중요한 조작체계중의 하나로 된다. 이 책에 있는 많은 UNIX실례들이 SVR 4에서 나온것이다.

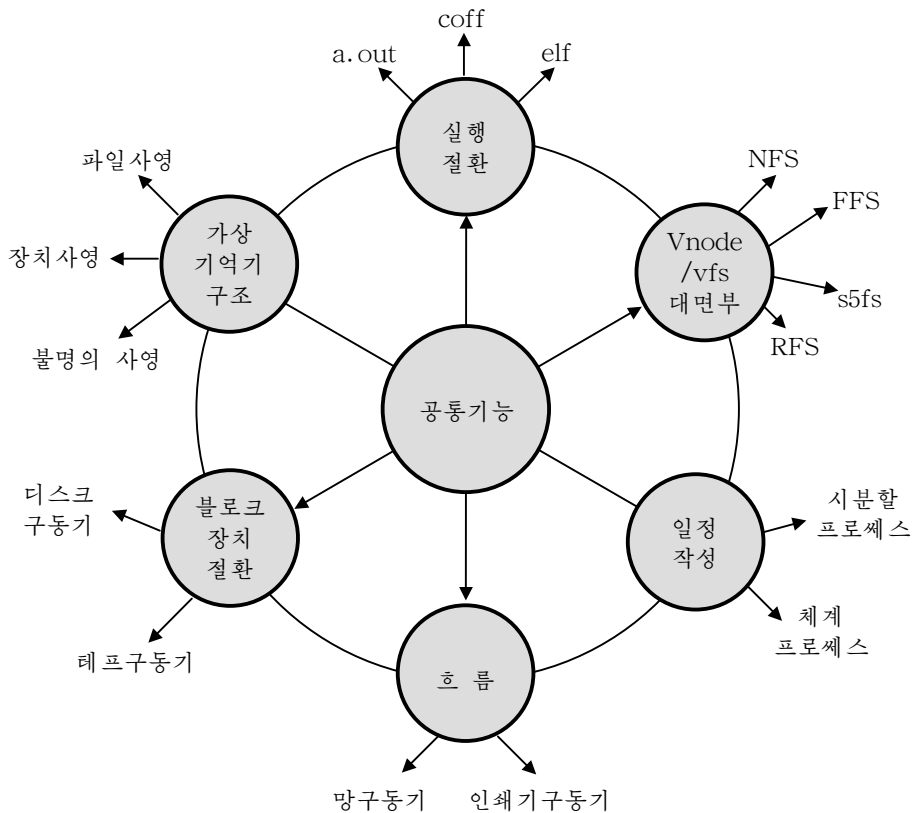



그림 2-17. 현대적인 UNIX 핵심부[VAHA96]

Solaris 2.x

Solaris는 Sun회사의 SVR 4에 기초한 UNIX판으로서 최근판본은 2.8로 되여 있다. Solaris의 판본 2 의 실현물은 SVR 4의 모든 기능들을 주며 그밖에 몇가지 보다 현대적인 기능들을 주고 있다. 그 기능들을 보면 충분히 선취가능하고 다중스레드화된 핵심부, SMP에 대한 충분한 지원, 파일체계에 의한 객체지향대면부들이다. Solaris는 가장 널리 쓰이고 있으며 가장 성공적인 상업적 UNIX실현물이다. 일부 OS기능에 대하여 Solaris는 이 책에 있는 UNIX실례들을 주고 있다.

4.4 BSD

Berkeley쏘프트웨어분산 BSD계렬의 UNIX판들은 OS설계리론의 개발에서 주요한 역할을 놀았다. 4.x BSD는 과학연구설비로서 널리 쓰이고 있으며 많은 상업적 UNIX제품들의 기초로 봉사해 왔다. BSD는 UNIX의 대중화에 대한 책임을 지고 있으며 UNIX에 대한 대부분의 개선은 우선 BSD판본들에서 나타났다고 말하면 아마 BSD에 대해 짐작할수 있을것이다.

4.4 BSD는 Berkeley가 출판하려고 하였던 BSD의 최종판본인데 설계와 실현에 대한 조직사업은 하지 않았다. 그것은 4.3 BSD에 대한 주되는 갱신판으로서 새로운 가상

기억기체계, 핵심부구조의 변경, 여러가지 다른 기능의 강화 등을 포함하고 있다.

Linux

개발과정

Linux는 IBM PC구성방식을 위한 UNIX방안으로서 출발하였다. 초기판본은 콤퓨터과학을 하는 핀란드의 대학생인 Linus Torvalds가 작성하였다. 그는 1991년에 Linux의 초기판본을 인터네트상에 발표하였다. 그때부터 인터네트상에서 합작하는 많은 사람들이 모두 Torvalds의 조종하에 Linux개발에 참가하였다. Linux는 자유롭게 원천코드를 사용할수 있었으므로 Sun Microsystems, Digital Equipment Corp(현재는 Compaq) 그리고 Silicon Graphics회사가 제공하는 모든 UNIX워크스테이션들을 거의 대신할수 있는 정도로 되였다. 오늘날 Linux는 이 모든 가동환경상에서 실행하는 충분한 기능을 가진 UNIX체계로 되였다.

Linux의 성공의 열쇠는 자유쏘프트웨어기금(FSF)의 주최하에 사용할수 있는 자유제품와 같은 그의 특질에 있었다. FSF의 목표는 자유롭고 질이 높은 안정하고 가동환경에 무관계한 쏘프트웨어이며 사용자대중을 기꺼이 받아 들이는데 있다. FSF의 GNU계획은 쏘프트웨어개발자들에게 도구를 주며 GNU공공허가증(GNU Public License(GPL))은 FSF의 승인표식이다. Torvalds는 자기의 핵심부개발에 GNU도구를 사용하였는데 이것은 그때 그가 GPL하에서 출판한것이다. 이와 같이 오늘날 보게 되는 Linux배포는 FSF의 GNU계획, Torvalds의 개별적인 노력과 세계적범위에서 많은 협력자들이 내놓은 산물이다.

많은 개별적인 프로그람작성자들이 그것을 사용하는외에 Linux는 현재 주식세계에 의의 있게 침투하였다[MANC00]. 이것은 초보적으로 자유쏘프트웨어이기때문인것이 아니라 Linux핵심부의 질때문이다. 많은 재능 있는 프로그람작성자들이 현재판본에 기여하여 기능수준이 높은 제품을 내놓고 있다. 더우기 Linux는 높은 수준에서 모듈화되여 있으며 쉽게 구성할수 있다. 이것은 여러가지 하드웨어가동환경으로부터 최적성능을 쉽게 달성할수 있게 한다. 게다가 원천코드를 사용할수 있으므로 판매자들은 특정한 요구를 만족시키도록 응용프로그람과 편의프로그람들을 작성할수 있다. 이 책전반에 걸쳐 Linux핵심부내부를 구체적으로 보여 주게 된다.

모듈식구조

대부분의 UNIX핵심부들은 단일핵심부이다. 단일핵심부는 하나의 큰 코드블로크안에 조작체계의 모든 기능들을 가상적으로 포함하고 있으며 그것은 단일주소공간을 가지고 단일프로쎄스처럼 실행하는것이라는데 대해 이미 지적하였다. 핵심부의 모든 기능요소들은 그의 내부자료구조와 루틴들모두에 모듈과 루틴들을 다시 련결하고 다시 설치하며 그 변화가 영향을 주기전에 재기동해야 한다. 결국 새로운 장치구동프로그람이나 파일체계기능을 추가하는것과 같은 어떤 수정을 가하기가 힘들다. 이 문제는 Linux에서 특히 심각하다. 그 개발이 전 지구적이며 독립적인 프로그람작성자들이 완만하게 련합된 그루빠로서 진행하기때문이다.

이 문제를 해결하기 위하여 **적재가능한 모듈**이라고 하는 상대적으로 독립인 블로크들의 집합으로 Linux를 조직한다[GDYE99]. Linux적재가능한 모듈은 두개의 중요한 특징을 가진다. 즉

- **동적련결 :** 핵심부가 이미 기억기내에 자리잡고 집행하고 있는 동안 핵심부모듈을 적재시켜 핵심부에 련결할수 있다. 모듈은 또한 임의의 시간에 기억기에서 분리시키고 제거할수 있다.
- **탄창가능모듈 :** 모듈들은 계층구조적으로 배치된다. 개별적모듈은 계층구조에서 보다 높은 의뢰기모듈이 참조할 때는 서고로 봉사하며 보다 낮은 모듈이 참조할 때는 의뢰기로 봉사한다.

동적련결[FRAN 97]은 구성과제를 쉽게 해주며 핵심부기억기를 절약한다. Linux에서 사용자프로그람 또는 사용자는 insmod와 rmmod지령을 사용하여 핵심부모듈을 명백히 표현되게 적재하거나 부리울수 있다. 핵심부자체는 특정한 기능에 대한 요구를 감시하며 요구에 따라 모듈을 적재하거나 부리울수 있다. 탄창가능모듈에 대한 모듈사이의 의존성을 정의할수 있다. 이것은 두가지 편리성을 가지게 한다. 즉

1. 류사한 모듈모임(실례로 류사한 하드웨어용구동기들)에 공통인 코드를 복사를 줄이면서 한개의 모듈에로 옮겨 놓을수 있다.
2. 핵심부는 다른 실행중의 모듈이 의거하고 있는 모듈부리우기를 그만두고 새로운 모듈이 적재될 때 추가적으로 요구되는 모듈을 적재하여 요구되는 모듈이 현재 있다는것을 확실하게 해줄수 있다.

그림 2-18은 모듈관리를 위해 Linux가 사용한 구조들을 례증하는 한 실례이다.이 그림은 단지 두개 모듈을 적재한후의 핵심부모듈목록을 보여 준다. 두개의 모듈은 FAT와 VFAT이다. 매개 모듈은 두개의 표 즉 모듈표와 기호표로 정의된다. 모듈표는 다음과 같은 요소들을 포함하고 있다. 즉

- next : 다음모듈에 대한 지시자. 모든 모듈은 어떤 련결목록으로 조직된다. 이 목록은 어떤 허위모듈로부터 시작한다(그림 2-18에서는 보여 주지 않았다.).
- ref : 이 모듈을 사용하는 모듈들의 목록
- symtab : 이 모듈의 기호표에 대한 지시자
- name : 모듈이름
- size : 기억기페지로 표시한 모듈크기
- addr : 모듈의 시작주소
- state : 모듈의 현 상태
- clean up() : 모듈부리우기에 착수하는 루틴을 가리킨다.

기호표는 그밖에 다른데서 사용하게 되는 이 모듈에 의하여 조종되는 기호들을 정의한다. 그것은 다음과 같은 요소들을 포함하고 있다. 즉

- size : 표의 총 크기
- n-symbols : 기호들의 개수
- n-refs : 참조의 회수
- symbols : 기호들의 표
- references : 이 모듈에 의존하는 모듈들의 목록

그림 2-18은 VFAT모듈이 FAT모듈다음에 적재되며 VFAT모듈이 FAT모듈에 의거하고 있다는것을 보여 주고 있다

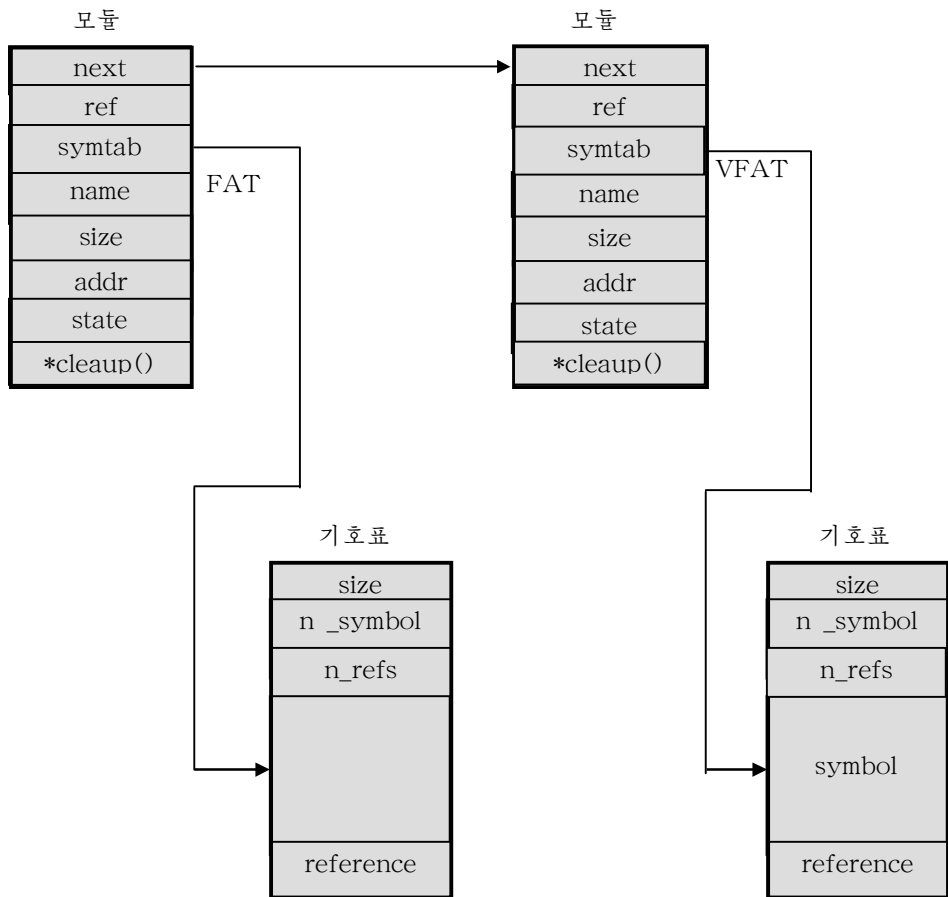



그림 2-18. Linux 핵심부모듈의 실례목록

참 고 문 헌

콤퓨터구성방식에서와 같이 조작체계에 대한 책들이 많다.[SILB00], [NUTT 00]과 [CROW97]은 연구실례들로서 많은 중요한 조작체계를 사용하는 기본원리를 포괄하고 있다.

몇가지 방안들에 대한 비교분석을 하면서 UNIX내부에 대하여 설명을 잘한것은 [VAHA96]이다. UNIX SVR 4에 대하여 [G00D94]는 충분한 기능적세부들로 명확하게 설명하고 있다. 과학용으로 대중화된 Berkeley UNIX 4.4 BSD에 대하여서는 [MCKU96]을 참고할수 있다. UNIX에 대한 중요한 론문묶음집이 1978년 7~8월에 Bell system Technical Journal의 특집으로 출판되였다. 이것들은 [AT & T87a와 b]로 재판되였다. Silicon 2.X를 간결하면서도 전망적으로 취급한것은 [GRAH95]이다.

Linux내부에 대해 잘 취급한것은 [BECK98]과 [CARD97]이다. 아직까지 Windows 2000의 내부에 대하여 집필된것은 많지 않다. [SOLO98]은 Windows NT내부를 잘 취급하였다.

ATT87a AT&T. *UNIX System Readings and Examples, Volume I.* Englewood Cliffs, NJ:
Prentice Hall, 1987.
ATT87b AT&T. *UNIX System Readings and Examples, Volume II.* Englewood Cliffs, NJ: Prentice Hall, 1987.
BECK98 Beck, M., et al. *Linux Kernel Internals.* Reading, MA: Addison-Wesley, 1998.
CARD97 Card, R.; Dumas, E.; and Mevel, F. *The Linux Kernel Book.* New York:Wiley, 1997.
CROW97 Crowley, C. *Operating Systems: A Design-Oriented Approach.* Chicago: Irwin, 1997.
GOOD94 Goodheart, B., and Cox, J. *The Magic Garden Explained: The Internals of UNIX System V Release 4.* Englewood Cliffs, NJ: Prentice Hall, 1994.
GRAH95 Graham, J. *Solaris 2.x: Internals and Architecture.* New York: McGraw-Hill, 1995.
MCKU96 McKusick, M.; Bostic, K.; Karels, M.; and Quartermain, J. *The Design and Implementation of the 4.4BSD UNIX Operating System.* Reading, MA: Addison-Wesley, 1996.
NUTTOO Nutt. G. *Operating Systems: A Modern Perspective.* Reading, MA: Addison-Wesley, 2000.
SILBOO Silberschatz, A.; Galvin, P.; and Gagne, G. *Applied Operating System Concepts.* Reading, MA: Addison-Wesley, 2000.
SOLO98 Solomon, D. *Inside Windows NT.* Redmond, WA: Microsoft Press, 1998.
VAHA96 Vahalia, U. *UNIX Internals: The New Frontiers.* Upper Saddle River, NJ: Prentice Hall, 1996.

련 습 문 제

1. 매개 일감이 동일한 특성들을 가지는 다중프로그람방식의 콤퓨터를 가지고 있다고 하자. 어떤 일감의 한개의 계산주기 T내에서 그 시간의 절반은 입출력에 소비되며 다른 절반은 처리기동작에 소비된다. 매개 일감은 총 N개의 주기동안 실행한다. 단순한 순환일정작성법을 사용하며 입출력조작은 처리기동작과 중복될수 있다고 가정하자. 이때 다음의 량들을 정의하시오.

   - 일감처리시간 = 어떤 일감을 완료하는데 걸리는 실제적시간
   - 처리능력 = 시간주기 T동안에 완료된 평균일감의 수
   - 처리기의 사용률 = 처리기가 동작(기다림이 아니라)하는 시간의 백분률

   한개,두개 그리고 네개의 동시적일감에 대하여 이 량들을 계산하시오. 이때 주기 T는 다음과 같은 방법으로 각각 분포된다고 가정한다.

   ㄱ)입출력에 첫번째 절반, 처리기에 두번째 절반
   ㄴ)입출력에 첫번째와 네번째 1/4, 처리기에 두번째와 세번째 1/4

2. 입출력위주의 프로그람은 혼자서 실행한다면 처리기를 사용하는것보다 보다 많은 입출력기다림시간을 소비하는 프로그람이다. 처리기위주의 프로그람은 그반대이다. 어떤 단기일정작성알고리듬이 방금전에 처리기시간을 조금 사용한 그런 프로그람들을 후원한다고 하자. 그러면 왜 이 알고리듬이 입출력위주의 프로그람을 후원하면서도 처리기위주의 프로그람에 대한 처리기시간을 영구적으로 거절하지 않는가에 대하여 설명하시오.

3. 어떤 콤퓨터가 캐쉬, 주기억기 그리고 가상기억기용으로 사용하는 디스크를 가지고 있다. 참조하는 단어가 캐쉬에 있으면 그것을 호출하는데 20ns가 걸린다. 그것이 캐쉬가 아니라 주기억기에 있다면 그것을 캐쉬에 적재하는데 60ns가 걸리며 참조를 다시 시작해야 한다. 그 단어가 주기억기에 없다면 그 단어를 디스크에서 불러 내는데 12ms가 걸리고 뒤이어 그것을 캐쉬에 복사하는데 60ns가 걸리며 그다음에 참조를 다시 시작한다. 캐쉬의 명중률은 0.9이며 주기억기의 명중률은 0.6이다. 이 체계에서 참조되는 단어에 접근하는데 ns로 평균 얼마만한 시간이 걸리는가?

4. 다중프로그람방식의 일괄처리를 최적화하는데 사용하는 방책들에 의하여 어떤 시분할체계를 최적화하려고 할 때 사용할수 있는 일정작성방법들을 대조하시오.

5. 체계호출의 목적은 무엇이며 체계호출은 조작체계와 그리고 2중방식(핵심부방식과 사용자방식)조작에 대한 개념과 어떻게 련관되여 있는가?

6. IBM의 대형콤퓨터의 조작체계 OS/390에서 핵심부의 주요한 모듈중의 하나가 체계자원관리자(System Resource Manager(SRM))이다. 이 모듈은 주소공간(프로쎄스들)사이에서 자원배정에 대한 책임을 지고 있다. SRM은 OS/390에 조작체계들중에서 유일하게 정교한 등급을 주고 있다. 그 어떤 다른 대형콤퓨터조작체계들과 확실하게는 그 어떤 다른 형태의 조작체계들도 SRM실제기억기와 입출력통로들을 포함한다. SRM은 처리기, 통로, 여러가지 주요자료구조의 사용과 관련한 통계자료를 축적한다. 그 목적은 성능감시와 분석에 기초하여 최적성능을 주는데 있다. 설치조작은 가치 있는 여러가지 성능의 객체들을 설정하며 이것들은 SRM에 대한 안내로 봉사한다. 그것은 체계사용률에 기초하여 설치조작과 일감성능특성들을 동적으로 변경한다. 또한 SRM은 숙련된 조작공이 사용자봉사를 개선하기 위해 구성 및 파라메터설정을 개선할수 있게 하는 보고를 제공한다.

이 문제는 SRM동작의 한 실례와 관련된다. 실제기억기는 프레임이라고 하는 동일한 크기의 블로크들로 분할된다. 그 프레임은 몇천개가 될수 있다. 매개 프레임은 페지라고 하는 어떤 가상기억기블로크를 유지하고 있다. SRM은 대략 초당 20회의 조종을 받으며 각각 매개 페지의 프레임을 조사한다. 페지가 참조되지 않았거나 변화되지 않았으면 어떤 계수기가 1만큼 증가된다. 시간에 따라 SRM는 이 수들을 평균하여 체계내의 페지프레임이 접촉되지 않은 평균시간을 결정한다. 이렇게 하는 목적은 무엇이며 SRM이 어떤 동작을 취할수 있는가?

제 2 편. 프로쎄스

제 2 편의 중심

모든 현대적인 조작체계의 기본과제는 프로쎄스관리이다. 조작체계는 자원을 프로쎄스에 할당하여 프로쎄스들이 정보를 공유하고 교환하며 매개 프로쎄스들의 자원을 다른 프로쎄스들로부터 보호할수 있게 해야 하며 프로쎄스들사이에서 동기화를 보장할수 있게 해야 한다. 이 요구를 만족시키기 위하여 조작체계는 매개 프로쎄스용자료구조를 유지해야 하는데 이 자료구조는 프로쎄스의 상태와 자원소유권을 서술하고 있으며 조작체계가 프로쎄스조종에 영향을 줄수 있게 한다.

다중프로그람처리방식의 단일처리기상에서 여러 프로쎄스들을 집행하는 과정은 시간적으로 교차될수 있다. 다중처리기상에서는 교대로 프로쎄스집행을 할수 있을뿐아니라 여러 프로쎄스들을 동시에 집행할수 있다. 교차식집행과 동시집행은 모두 병행성의 형태로서 응용프로그람작성자들과 조작체계에 중요하면서도 어려운 문제들을 제기한다.

많은 당시의 조작체계들에서 프로쎄스관리의 난점들이 스레드개념의 도입으로 혼탕되여 있다. 다중스레드처리체계에서 프로쎄스는 자원소유관계에 대한 속성을 유지하는 한편 다중, 병행집행흐름에 대한 속성은 프로쎄스에서 실행하는 스레드의 특성으로 된다.

제 2 편의 안내

제 3 장. 프로쎄스의 서술과 조종

전통적인 조작체계의 초점은 프로쎄스관리에 있다. 각 프로쎄스는 임의의 시간에 준비, 실행 및 폐색과 같은 많은 집행상태들중의 어느 하나에 있게 된다. 조작체계는 이 집행상태들에 대한 추적을 보존하며 그 상태들사이에서 프로쎄스의 움직임을 관리한다. 이 목적으로부터 조작체계는 매개 프로쎄스를 서술하는 자료구조를 유지한다기보다 정교하게 만든다. 조작체계는 일정작성기능을 수행해야 하며 프로쎄스공유와 동기화를 위한 기능을 제공해야 한다. 제3장에서는 프로쎄스관리를 위한 대표적인 조작체계들에서 사용된 자료구조와 수법을 고찰한다.

제 4 장. 스레드, SMP 및 마이크로핵심부

제4장에서는 많은 당시의 조작체계들을 특징 지어 주며 전통적인 조작체계설계의 우점들을 보여 주는 세가지 령역들을 포괄하고 있다. 많은 조작체계들에서 프로쎄스에 대한 고전적인 개념은 두개 부분으로 나누어 져 있었다. 즉 하나는 자원소유권(프로쎄스)을 취급하며 하나는 명령집행흐름(스레드)을 취급한다. 한개의 프로쎄스는 여러개의 스레드를 포함할수 있다. 다중스레드식구성은 응용프로그람의 구조화에서나 성능측면에서

나 다 우점을 가진다. 제4장은 또한 대칭형다중처리기(SMP)를 론의하는데 이것은 여러개의 처리기를 가진 콤퓨터로서 처리기들은 각기 모든 응용프로그람과 체계코드를 집행할수 있다. SMP구성은 성능과 믿음성을 증가시킨다. SMP는 흔히 다중스레드처리와 결합하여 쓰이지만 다중스레드처리를 하지 않을 때에도 강력한 개선을 얻을수 있다. 끝으로 제4장에서는 마이크로핵심부를 론의하는데 이것은 조작체계설계의 한 류형이다. 이 류형은 핵심부방식으로 실행하는 조작체계코드의 량을 최소화한다. 이 방법의 우점을 분석한다.

제 5 장. 병행성: 호상배제와 동기화

현대적인 조작체계의 두가지 중심적인 주제는 다중프로그람처리와 분산처리이다. 이 두가지 주제에서 기본적이며 조작체계설계기술에서 기본적인것은 병행성이다. 제5장에서는 병행성의 두가지 측면 즉 호상배제와 동기화를 고찰한다. 호상배제는 단지 한개의 프로쎄스만이 어떤 시간에 공유된 객체에 접근하는 방법으로 코드, 자원이나 자료를 공유하는 다중프로쎄스들(스레드들)의 능력이다. 호상배제와 관련된것이 동기화인데 이것은 정보를 교환할 때 그것들의 동작을 일치시키는 다중프로쎄스들의 능력이다. 제5장에서는 제기된 설계문제들에 대한 설명으로부터 시작하여 병행성과 관련된 내용들을 많이 취급하고 있다. 이 장에서는 병행성에서의 하드웨어지원에 대하여 설명하고 그다음 병행성 즉 신호기, 감시기, 통보문넘기기를 지원하는 가장 중요한 수법들을 고찰한다.

제 6 장. 병행성: 교착과 고갈

제6장에서는 병행성에 대한 두가지 추가적인 측면을 고찰한다. 교착은 두개 또는 그 이상의 프로쎄스들의 모임이 작업을 계속하기 위하여 그 모임에 속한 다른 성원들로 하여금 어떤 조작을 완료하기를 기다리고 있지만 어느 성원도 작업을 계속할수 없는 상황을 가리키는 말이다. 교착은 예상하기 힘든 현상으로서 이 문제에 대한 그 어떤 쉽고도 일반적인 해결방도는 없다. 제6장에서는 교착을 취급하는 세개의 중요한 방법 즉 예방, 피하기, 검출에 대하여 고찰한다. 고갈은 프로쎄스가 집행할 준비는 되여 있지만 다른 프로쎄스들에 밀리면서 처리기에 대한 접근을 계속 무시 당하는 상황을 가리키는 말이다. 고갈은 일정작성문제로서 편을 정하여 취급할 내용으로 보고 제4편에서 취급한다. 제6장에서는 교착에 중심을 두고 있지만 교착해결방도가 고갈문제의 피하기를 요구하므로 고갈에 대해서도 설명한다.

제 3 장. 프로쎄스의 서술과 조종

조작체계설계는 일정한 일반적요구들을 반영하여야 한다. Windows 98과 같은 단일 사용자체계로부터 수천명의 사용자를 지원할수 있는 OS/390과 같은 일반형콤퓨터체계에 이르기까지 모든 다중프로그람식조작체계는 프로쎄스에 대한 개념으로 만들어 지고 있다. 그러므로 조작체계가 만족시켜야 할 대부분의 요구는 프로쎄스들을 참조하여 표현할수 있다. 즉

- 조작체계는 여러 프로쎄스들의 집행을 교차처리하면서 처리기의 사용을 최소화해야 하며 한편 해당한 응답시간을 보장해야 한다.
- 조작체계는 특정한 방법에 따라(실례로 어떤 기능이나 응용프로그람이 보다 높은 우선권을 가질수 있다.) 처리하도록 자원을 배정해야 하며 한편 같은 시각에 교착을 피하여야 한다.[1]
- 프로쎄스사이 통신과 사용자가 프로쎄스를 창조하는것을 지원하기 위해 조작체계를 요구할수 있는데 그것들은 모두 응용프로그람의 구조화를 방조할수 있다.

여기서는 조작체계에 대한 구체적인 연구를 조작체계가 프로쎄스를 표현하고 조종하는 방법에 대한 설명으로부터 시작한다. 우선 프로쎄스의 상태를 론의하는데 이것은 프로쎄스의 성질을 특징 짓는다. 그다음 매개 프로쎄스를 표현하기 위하여 조작체계가 요구하는 자료구조와 조작체계가 자기의 객체들을 얻기 위해 요구하는 프로쎄스의 다른 기능들을 고찰한다. 끝으로 UNIX SVR 4에서 프로쎄스의 관리에 대하여 론의한다.

주의 : 이 장에서 때때로 가상기억기를 참조하게 된다. 지금까지는 프로쎄스를 취급하는데서 이 개념을 무시할수 있었다. 그러나 일정한 설명을 할 때 가상기억기를 적절히 고찰하여야 한다. 가상기억기를 제8장까지는 세부적으로 고찰하지 않지만 제2장에서 주요개괄을 주고 있다.

제 1 절. 프로쎄스의 상태

처리기의 기본기능은 주기억기에 상주하는 기계명령을 집행하는것이다. 이 명령은 프로그람형태로 주어 진다. 앞의 장들에서 설명한바와 같이 프로그람작성의 효과성과 용이성을 보장하기 위해 처리기는 많은 프로그람집행을 시간적으로 교차처리할수 있다.

제2장에서 언급한바와 같이 프로그람이 집행되자면 프로그람을 위한 프로쎄스나 과제가 창조되여야 한다. 처리기의 견지에서 볼 때 그것은 프로그람계수등록기안의 값들을 변화시켜 얻어 지는 일정한 순서에 따라 그것의 저장소에서 명령들을 불러 내여 집행한다. 시간이 흐름에 따라 프로그람계수기는 서로 다른 프로쎄스들의 일부분으로 되는 각이한 프로그람내의 코드를 가리킬수 있다. 개별적프로그람의 견지에서 그의 집행은 프로그람내에서 명령의 순서를 포함하고 있다.

---

1. 교착은 제6장에서 론의한다. 교착은 본질적으로 두 프로쎄스가 같은 두개의 자원을 요구할 때 생긴다. 각자는 그에 대한 소유권을 가지고 있다. 각 프로쎄스가 자원을 차지하지 못한채 무한히 기다릴수 있다.

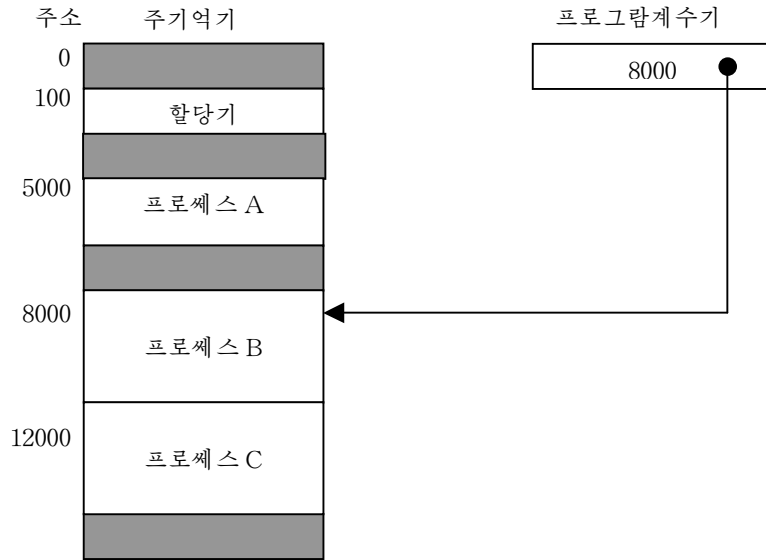



그림 3-1. 명령주기 13에서 실례의 집행(그림 3-3)에 대한 순서상태

프로쎄스를 집행하는 명령의 순서를 목록화하여 개별적프로쎄스들의 성질을 특징 지을수 있다. 그러한 목록작성을 프로쎄스의 **추적**이라고 한다. 여러가지 프로쎄스들의 추적이 어떻게 교차처리되는가를 보고 처리기의 성질을 특징 지을수 있다 .

매우 간단한 실례를 고찰해 보자. 그림 3-1은 세개 프로쎄스의 기억기배치도를 보여주고 있다. 설명을 간단히 하기 위하여 가상기억기를 전혀 사용하지 않는다고 가정한다. 때문에 세개의 프로쎄스는 모두 주기억기내에 완전히 적재되여 있는 프로그람들이라고 할수 있다. 그밖에 한 프로쎄스에서 다른 프로쎄스에로 처리기를 절환하는 작은 할당기 프로그람이 있다. 그림 3-2는 집행초기의 세 프로쎄스에 대한 추적들을 보여 주고 있다.

ㄱ)	ㄴ)	ㄷ)
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = 프로쎄스 A의 프로그람시작주소
8000 = 프로쎄스 B의 프로그람시작주소
12000 = 프로쎄스 C의 프로그람시작주소

그림 3-2. 그림 3-1의 프로쎄스들의 추적
ㄱ-프로쎄스 A의 추적, ㄴ-프로쎄스 B의 추적, ㄷ-프로쎄스 C의 추적

프로쎄스 A와 C에서 집행한 첫 12개의 명령을 보여 주고 있다. 프로쎄스 B는 4개의 명령을 집행하며 이때 네번째 명령의 프로쎄스가 기다려야 할 입출력조작을 기동시킨다고 가정한다.

이제 처리기의 견지에서 이 추적들을 보자. 그림 3-3은 첫 52개의 명령주기(편의상 명령주기에 번호를 붙이고 있다.)에서 얻어 지는 교차처리된 추적을 보여 주고 있다. 여기서는 조작체계가 하나의 프로쎄스로 하여금 최대 6개의 명령주기동안만 집행을 계속할 수 있게 한다. 그후에 그것은 새치기를 받는데 이것은 임의의 단일프로쎄스가 처리기시간을 독점하는것을 예방한다. 그림 3-3에서 보여 준바와 같이 프로쎄스 A의 첫 6개의 명령은 집행되고 뒤이어 시간만기가 일어 나며 할당기에서 일부 코드를 집행하는데 6개

1 5000
2 5001
3 5002
4 5003
5 5004
6 5005
--------------------- 시간만기
7 100
8 101
9 102
10 103
11 104
12 105
13 8000
14 8001
15 8002
16 8003
-------------------- 입출력요청
17 100
18 101
19 102
20 103
21 104
22 105
23 12000
24 12001
25 12002
26 12003
27 12004
28 12005
---------------------시간만기
29 100
30 101
31 102
32 103
33 104
34 105
35 5006
36 5007
37 5008
38 5009
39 5010
40 5011
---------------------시간만기
41 100
42 101
43 102
44 103
45 104
46 105
47 12006
48 12007
49 12008
50 12009
51 12010
52 12011
----------------------시간만기

H

100 = 할당기프로그람의 시작주소.

어두운 구역은 할당기프로쎄스의 집행을 표시한다;
첫번째와 세번째 렬은 명령주기를 계수한다;
두번째와 네번째 렬은 집행되는 명령주소를 보여 주고 있다.

그림 3-3. 그림 3-1 의 프로쎄스들에 대한 결합식추적

의 명령을 집행한 다음에 할당기의 조종을 프로쎄스 B에 넘기기전에 6개의 명령을 집행한다. [2] 4개 명령을 집행한후에 프로쎄스 B는 기다려야 할 입출력동작을 요청한다. 따라서 처리기는 프로쎄스 B의 집행을 정지시키며 할당기를 거쳐 프로쎄스 C에로 옮겨 간다. 새치기후에 처리기는 프로쎄스 A에 돌아 간다. 이 프로쎄스가 시간만기될 때 프로쎄스 B는 입출력조작이 완료되기를 기다리고 있으며 그래서 할당기는 프로쎄스 C에 다시 옮겨 간다.

2상태프로쎄스의 모형

조작체계의 기본은 프로쎄스의 집행을 조종하는것으로서 이것은 집행에서 교차처리형태를 결정하며 프로쎄스에 자원을 할당하는 내용들을 포함하고 있다. 프로쎄스를 조종하는 프로그람설계에서의 첫 단계는 출품하려는 프로쎄스의 성질을 서술하는것이다.

여기서는 임의의 시간에 프로쎄스가 처리기에 의해 집행되는가 아닌가 하는것을 관찰하여 가장 간단하면서 가능한 모형을 구성할수 있다. 이로부터 프로쎄스를 두 상태 즉 그림 3-4 ㄱ에서 보여 준바와 같이 실행 또는 비실행상태중의 하나에 있게 할수 있다. 조작체계는 새로운 프로쎄스를 창조할 때 그 프로쎄스를 비실행상태에 있는 체계에 넣는다. 조작체계에 알려 져 있으며 집행할 기회를 기다리는 프로쎄스가 있다. 시간이 감에 따라 현재 실행하고 있는 프로쎄스는 새치기를 받게 될것이며 조작체계의 할당기부분은 새로운 프로쎄스를 선택하여 실행하게 된다. 전자의 프로쎄스를 실행상태로부터 비실행상태로 옮기고 다른 프로쎄스들중의 하나를 실행상태로 옮겨 놓는다.

이 간단한 모형으로부터 조작체계의 일부 설계요소들에 대한 평가를 할수 있다. 매개 프로쎄스들을 일정한 방법으로 표현하여 조작체계가 그에 대한 추적을 유지할수 있게 해야 한다. 즉 현 상태와 기억기에서의 위치를 포함하여 매개 프로쎄스와 관련되는 일정한 정보가 있어야 한다. 실행중이 아닌 프로쎄스들은 어떤 대기렬에 들어 가 자기의 집행차례를 기다려야 한다. 그림 3-4 ㄴ는 그와 관련한 구조를 제기하고 있다. 단일대기렬이 있는데 여기에서 매개 입구점은 특정한 프로쎄스에 대한 지적자로 된다. 또한 대기렬을 변경된 자료블로크의 목록으로 구성할수 있으며 여기서 매개 블로크는 하나의 프로쎄스를 표시한다. 후에 이 후자의 실현문제를 연구하게 된다.

이 대기선도를 사용하여 할당기의 성질을 서술할수 있다. 새치기를 받은 프로쎄스는 기다리고 있는 프로쎄스들의 대기렬에 이송된다. 또한 프로쎄스가 완료되거나 취소되면 그것을 버린다(체계를 출구한다.). 어느 경우든 할당기는 대기렬에서 프로쎄스를 선택하여 집행한다.

프로쎄스의 창조와 완료

2상태모형을 고찰하기전에 프로쎄스의 창조와 완료에 대하여 론의한다. 극단적으로 사용되는 프로쎄스모형의 성질에 관계없이 프로쎄스의 수명은 그의 창조와 완료에 의해 정해 진다.

프로쎄스의 창조

새로운 프로쎄스를 현재 관리하고 있는것에 추가하려면 조작체계는 프로쎄스를 관리하는데 사용되는 자료구조를 만들며(제3장 제2절에 서술되여 있다.) 그 프로쎄스에 주기억기의 주소공간을 배정한다. 이 동작들에 의하여 새로운 프로쎄스가 창조된다.

표 3-1에서 지적된바와 같이 4개의 공통적인 사건들이 프로쎄스를 창조한다. 일괄

2. 프로쎄스와 할당기가 집행한 작은 몇개의 명령들은 비현실적이며 저급하다. 설명을 명백하게 하기 위하여 간단화된 실례로서 사용되고 있다.

환경에서 프로쎄스는 일감에 의뢰하여 창조된다. 대화형환경에서 프로쎄스는 새로운 사용자가 가입하려고 할 때 창조된다. 두 경우에 모두 조작체계는 새로운 프로쎄스의 창조에 응답할수 있다. 조작체계는 또한 응용프로그람을 대신하여 프로쎄스를 창조할수 있다. 실례로 사용자가 파일을 인쇄하려고 한다면 조작체계는 인쇄를 관리하게 될 프로쎄스를 창조할수 있다. 프로쎄스를 요구하는것은 이처럼 인쇄과제를 완료하는데 요구되는 시간에 무관계하게 생겨 날수 있다.

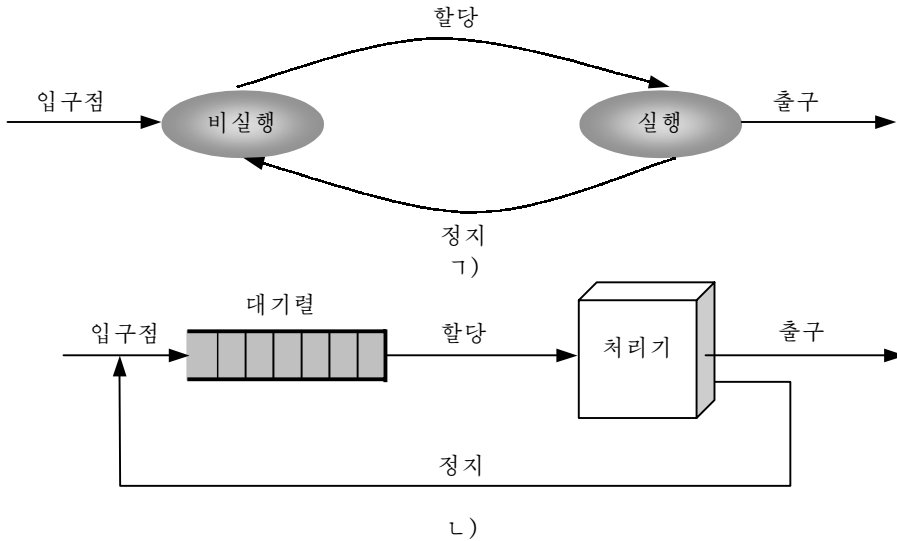



그림 3-4. 2상태프로쎄스의 모형
ㄱ-상태이행도, ㄴ-대기선도

전통적으로 조작체계는 사용자 즉 응용프로그람에 투명한 방법으로 모든 프로쎄스들을 창조하였고 이것은 아직도 많은 조작체계들에서 공통적으로 찾아 보게 된다. 그러나 하나의 프로쎄스가 다른 프로쎄스를 창조할수 있게 하는것이 유용할수 있다. 실례로 응용프로쎄스가 다른 프로쎄스를 생성하여 응용프로그람이 생성하는 자료를 수신하거나 후에 분석하는데 알맞는 형태로 자료를 조직할수 있다. 새로운 프로쎄스는 응용프로그람을 병렬로 실행하며 새로운 자료가 나타날 때마다 활성화된다. 이런 배렬은 응용프로그람을 구조화하는데서 매우 유용할수 있다. 다른 실례에서 봉사기프로쎄스(실례로 봉사기, 파일봉사기)가 그것이 조종하는 매개 요청에 따라 새로운 프로쎄스를 생성할수 있다. 조작체계가 또 다른 프로쎄스의 요청에 따라 어떤 프로쎄스를 창조할 때 그 동작을 프로쎄스새끼치기라고 한다.

표 3-1. 프로쎄스창조의 리유

새로운 일괄일감	조작체계는 보통 테프나 디스크상에 일괄일감조종흐름으로 주어 진다. 새로운 일감을 접수할수 있게 조작체계가 준비되면 그것은 다음순서의 일감조종지령을 읽어 들이게 된다.
대화형가입	말단에 있는 사용자가 체계에 가입한다.
봉사를 위해 OS가 창조한다.	조작체계는 프로쎄스를 창조함으로써 사용자프로그람을 대신하여 사용자를 기다리게 하지 않고 어떤 기능을 수행하게 할수 있다(실례로 인쇄를 조종하기 위한 프로쎄스).
현존프로쎄스들이 새끼를 친다.	모듈화 또는 병렬화를 위해 사용자프로그람이 몇개의 프로쎄스들을 창조하도록 명령할수 있다.

한 프로쎄스가 다른 프로쎄스를 새끼칠 때 전자를 부모프로쎄스라고 하며 새끼친 프로쎄스를 자식프로쎄스라고 한다. 대체로 《관련 있는》 프로쎄스들은 서로 통신하며 협동동작할것을 요구한다. 이 협동동작을 실현하는것은 프로그람작성자에게 있어서 힘든 과제이다. 이 문제는 제5장에서 론의한다.

프로쎄스의 완료

표 3-2는 프로쎄스완료의 대표적인 리유들을 요약하고 있다. 콤퓨터체계는 프로쎄스가 자기의 완료를 알려 줄수 있는 수단을 보장해야 한다. 일괄일감은 정지명령이든가 완료하기 위한 명백한 조작체계봉사호출을 포함하고 있어야 한다. 전자의 경우에 정지명령은 새치기를 발생시켜 프로쎄스를 완료한 조작체계에 경보를 줄수 있다. 대화형응용프로그람에서 사용자의 동작이 언제 프로쎄스가 완료되는가를 알려 주게 된다. 실례로 시분할체계에서 특정한 사용자를 위한 프로쎄스는 사용자가 탈퇴를 끝내거나 자기말단의 전원을 차단시킬 때 완료된다. 개인용콤퓨터나 워크스테이션에서 사용자는 응용프로그람을 포기할수 있다(실례로 단어처리나 자료표). 이 모든 동작들은 조작체계에 봉사요청을 발생시켜 요청하는 프로쎄스를 완료하게 한다.

표 3-2. 프로쎄스의 완료리유

정상완료	프로쎄스는 OS봉사호출을 집행하여 실행을 완료했다는것을 알려 준다.
시간한계초과	프로쎄스가 특정한 총적시간한계보다 훨씬 오래 실행하였다. 측정되는 시간형에 몇가지 가능성이 있다. 이것들은 총 경과시간, 집행하는데 소비한 시간, 대화형프로쎄스인 경우 사용자가 임의의 입력을 한 때로부터의 시간들을 포함하고 있다.
기억기사용불가능	프로쎄스가 체계가 줄수 있는것보다 더 많은 기억기를 요구한다.
경계위반	프로쎄스가 접근을 허용하지 않는 기억기위치에 접근하려고 한다.
보호오유	프로쎄스가 사용을 허용하지 않는 자원이나 파일을 사용하려고 하거나 읽기전용파일을 쓰는것과 같은 부정확한 방식으로 사용하려고 한다.
산수연산오유	프로쎄스가 0으로의 나누기와 같은 금지된 연산을 하려고 하거나 하드웨어가 수용할수 있는것보다 더 큰 수를 기억하려고 한다.
시간과도실행	프로쎄스가 일정한 사건이 발생하는것을 특정한 최대시간보다 더 오래 기다렸다.
입출력실패	파일찾기에서 불가능성, 특정한 최대수의 시도후에 읽거나 쓰기실패(실례로 테프상의 결함이 있는 구역과 맞다들릴 때) 또는 무효조작(행인쇄기로부터의 읽기와 같은)과 같은 입력 또는 출력기간에 오유가 발생한다.
무효명령	프로쎄스가 존재하지 않는 명령을 실행하려고 한다(흔히 자료구역에로 이행하거나 자료를 집행하려고 할 때 생긴다.).
특권명령	프로쎄스가 조작체계에 예약된 명령을 사용하려고 한다. 자료쪼각이 틀린 형이거나 초기화되여 있지 않다.
조작자 또는 OS	일정한 리유로 하여 조작자 또는 조작체계가 프로쎄스를 완료하였다(실례로 교착이 존재한다면).
부모완료	부모가 완료할 때 조작체계는 자동적으로 그 부모의 모든 자식들을 완료한다.
부모요청	부모프로쎄스는 대체로 임의의 자기자식들을 완료할수 있는 권한을 가진다.

보충적으로 몇가지 오유와 고장조건들이 프로쎄스를 완료에로 이끌어 갈수 있다. 표 3-2는 보다 공통적으로 인정된 일부 조건들을 목록화하고 있다.[3]

끌으로 일부 조작체계들에서 그것을 창조하였거나 부모프로쎄스가 자기자체를 완료했을 때 바로 그 프로쎄스가 어떤 프로쎄스를 완료할수 있다.

5상태의 모형

모든 프로쎄스가 항상 집행준비가 되였다면 그림 3-4 ㄴ가 제기한 대기렬규칙이 효과적일것이다. 대기렬은 선입선출목록이며 처리기는 사용할수 있는 프로쎄스형태에서 **순환**방식으로 조작한다(대기렬에 있는 매개 프로쎄스들은 일정한 시간을 받으며 차례로 집행하고 페색되지 않는 한 대기렬에로 돌아 간다.). 그러나 여기서 서술한 간단한 실례에서 이 실현은 불가능하다. 즉 비실행상태에 있는 일부 프로쎄스들은 집행할 준비가 되여있고 한편 다른것들은 입출력조작이 완료되기를 기다리면서 페색된다. 이로부터 단일대기렬을 쓰면 할당기가 대기렬의 제일 오래된 끝에 있는 프로쎄스를 꼭 선택하지 못할수 있다. 할당기가 페색되지 않고 대기렬에 제일 오래 머물러 있은 프로쎄스를 찾아 그 목록을 주사하게 하는편이 더 합리적일수 있다.

이 상황을 조종하는 보다 자연적인 방법은 비실행상태를 두개의 상태 즉 준비 및 페색으로 나누는것이다. 이것을 그림 3-5에 보여 주고 있다. 효과적인 대책으로서 두개의 추가적인 상태를 보충하였는데 그 유용성이 증명되였다. 이 새로운 도해에서 5개의 상태는 다음과 같다. 즉

- **실행** : 현재 집행되고 있는 처리. 이 장에서는 단일처리기를 가진 콤퓨터를 가정하며 그런데로부터 대체로 한 시각에 이 상태에는 하나의 프로쎄스가 있을수 있다.
- **준비** : 기회가 차례지면 집행할 준비가 되여 있는 프로쎄스
- **페색** : 입출력조작의 완료와 같은 일정한 사건이 발생할 때까지 집행할수 없는 프로쎄스
- **새것** : 금방 창조되였지만 조작체계가 집행할수 있는 프로쎄스의 집결소에 가도록 아직 허가를 받지 못한 프로쎄스
- **출구** : 정지시켰거나 일정한 리유로 취소시켰기때문에 조작체계가 집행할수 있는 프로쎄스집결소로부터 해방된 프로쎄스

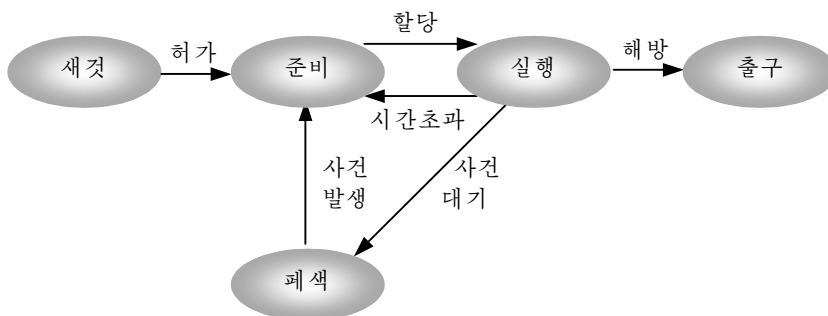



그림 3-5. 5 상태프로쎄스의 모형

[3] 어떤 관대한 조작체계는 일부 경우에 사용자로 하여금 프로쎄스를 완료하지 않고 고장을 회복할수 있게 해준다. 실례로 사용자가 파일에 대한 접근을 요구한다면 그 접근을 무시하고 조작체계는 프로쎄스가 계속 집행해 나가도록 한다는것을 통지할수 있다.

새것과 출구상태는 프로쎄스관리에서 유용한 구성물이다. 새것의 상태는 방금 정의된 프로쎄스에 대응한다. 실례로 새로운 사용자가 시분할체계에 가입하려고 하거나 새로운 일괄일감이 집행을 의뢰 받으면 조작체계는 두 단계로서 새로운 프로쎄스를 정의할수 있다. 우선 조작체계는 프로쎄스를 만드는데 필요한 정리조작들을 수행한다. 프로쎄스를 관리하는데 필요한 어떤 표들을 할당하고 만든다. 이 시점에서 프로쎄스는 새것상태에 있다. 이것은 조작체계가 프로쎄스를 창조하는데 필요한 동작을 수행했지만 그자체에 프로쎄스를 집행할데 대한 위임은 하지 않았다는것을 의미한다. 실례로 조작체계는 체계에 있을수 있는 몇개의 프로그람을 성능이나 주기억기한계때문에 제한할수 있다. 프로쎄스가 새것상태에 있는 동안 조작체계가 요구하는 프로쎄스와 관련 있는 정보는 주기억기에 있는 조종표에 유지된다. 그러나 프로쎄스자체는 주기억기에 없다. 즉 집행될 프로그람코드는 주기억기에 없으며 프로그람과 관련된 자료에 아무런 공간도 배정되여 있지 않다. 프로쎄스가 새것상태에 있는 동안 프로그람은 2차기억기 대체로 디스크기억기에 있다.[4]

류사하게 프로쎄스는 두 단계로 체계를 출구한다. 우선 본질적인 완료점에 도달했을 때, 회복할수 없는 오유가 있기때문에 취소시킬 때 또는 해당 권한을 가진 다른 프로쎄스가 그 프로쎄스를 취소시키려고 할 때 프로쎄스를 완료시킨다. 완료는 프로쎄스를 출구상태에로 이동시킨다. 이 시점에서 프로쎄스는 더이상 집행에 뽑힐 자격을 가지지 못한다. 일감과 관련된 표와 다른 정보는 림시로 조작체계가 보관하며 조작체계는 보조기억기나 지원프로그람이 임의의 요구되는 정보를 추출할수 있는 시간을 준다. 실례로 합계프로그람이 료금계산을 목적으로 처리기시간과 사용한 다른 자원들을 기록할것을 요구할수 있다. 편의프로그람이 성능이나 사용률분석과 관련하여 프로쎄스의 경력에 대한 정보를 뽑아 낼것을 요구할수 있다. 일단 프로그람경력에 대한 정보를 뽑아 내기만 하면 조작체계는 프로쎄스와 관련된 임의의 자료를 더이상 유지할 필요가 없으며 프로쎄스는 체계에서 삭제된다.

그림 3-5는 프로쎄스에서 매개 상태이행을 초래하는 사건의 형태들을 보여 주고 있다. 가능한 이행은 다음과 같다. 즉

- **빔→새것** : 프로그람을 집행하기 위하여 새로운 프로쎄스가 창조된다. 이 사건은 표 3-1에 목록화되여 있는 리유들중 몇가지로 인하여 발생한다.
- **새것→준비** : 추가적인 프로쎄스를 맡을 준비가 되면 조작체계는 프로쎄스를 새것상태로부터 준비상태로 이동시킨다. 대부분의 체계들은 몇개의 현존프로쎄스들이나 현존프로쎄스들에 위임한 가상기억기량에 기초하여 일정한 제한을 설정한다. 이 제한은 많은 능동프로쎄스들이 성능을 저하시키지 않도록 보증한다.
- **준비→실행** : 새로운 프로쎄스를 실행시키기 위하여 선택할 시간이 되면 조작체계는 준비상태에 있는 프로쎄스들중에서 하나를 선택한다. 어느 프로쎄스를 선택하는가 하는 문제는 제4편에서 설명한다.
- **실행→출구** : 프로쎄스가 완료했다는것을 알려 주든가 취소되면 조작체계는 현재 실행중인 프로쎄스를 완료한다.
- **실행→준비** :이행의 가장 공통적인 리유는 실행중인 프로쎄스가 새치기 받지 않은 집행에서 최대허용시간에 도달했다는것이다. 실제상 다중프로그람처리조작체계는 이런 류형의 시간규칙을 적용하고 있다. 이러한 이행을 일으키는 몇가지 다른 원인들이 있는데 모든 조작체계들에서 실현되는것은 아니다. 실례로 조작체계가 각

[4] 이 단락의 설명에서는 가상기억기에 대한 개념을 무시하고 있다. 가상기억기를 지원하는 체계들에서 프로쎄스가 새것으로부터 준비에로 넘어 갈 때 그것의 프로그람코드와 자료는 가상기억기에 적재된다. 가상기억기는 주로 제 2 장에서 설명하고 있으며 세부적인것은 제 8 장에서 설명하고 있다.

이한 준위의 우선권을 서로 다른 프로쎄스들에 설정한다면 어떤 프로쎄스를 선취할수 있다. 실례로 프로쎄스 A가 주어 진 우선권준위에서 실행중이고 프로쎄스 B가 기다리고 있는 사건이 발생했다는것을 조작체계가 알면 B를 준비상태로 옮긴 다음 프로쎄스 A를 새치기하여 프로쎄스 B를 할당할수 있다. 그것을 조작체계가 프로쎄스 A를 **선취**했다고 말한다.[5] 최종적으로 프로쎄스는 처리기의 조종을 자발적으로 해방시킬수 있다.

- **실행-페색** : 프로쎄스가 기다려야 할 어떤것을 요청하면 페색상태에 들어 간다. 조작체계에 대한 요청은 보통 체계봉사호출 즉 실행중인 프로그람으로부터 조작체계코드의 일부분인 수속에로 이행할것을 호출하는 형태로 된다. 실례로 프로쎄스는 조작체계가 즉시 수행하려고 준비하고 있지 않는 봉사를 조작체계에 요청할수 있다. 그것은 즉시에 사용할수 없는 가상기억기의 파일 또는 공유구간과 같은 자원을 요청할수 있다. 또는 프로쎄스가 입출력조작과 같은 동작을 초기화할수 있는데 그것은 그 프로쎄스를 계속해나가기전에 완료되여야 한다. 프로쎄스들이 서로 통신할 때 어떤 프로쎄스는 그것이 다른 프로쎄스가 입력을 제공하기를 기다리든가 또는 다른 프로쎄스로부터 통보문을 기다린다면 페색될수도 있다.
- **페색-준비** : 기다리던 사건이 발생하면 프로쎄스는 페색상태에서 준비상태로 이동된다.
- **준비-출구** : 명백하기때문에 상태도에서 이 이행은 보여 주지 않고 있다. 일부 체계들에서 부모가 자식프로쎄스를 임의의 시간에 완료시킬수 있다. 또한 부모를 완료하면 그 부모와 관련 있는 모든 자식프로쎄스들을 완료할수 있다.
- **페색-출구** : 선행한 항목들에 있는 설명을 적용하면 된다.

앞의 간단한 실례로 돌아 가면 그림 3-6은 매개 프로쎄스들이 상태들사이에서 움직이는 관계를 보여 주고 있다. 그림 3-7 ㄱ는 대기규칙을 실현할수 있는 방도를 제기하고 있다. 지금 2개의 대기렬 즉 준비대기렬과 페색대기렬이 있다. 매개 프로쎄스가 체계에 수용되면 그것은 준비대기렬에 배치된다. 조작체계가 실행할 또 다른 프로쎄스를 선택할 시간이 되면 그것은 준비대기렬중에서 하나를 선택한다. 아무런 우선권방안도 없다면 이것은 단순한 선입선출대기렬로 될수 있다. 실행중인 프로쎄스를 집행에서 제거할 때 그것은 완료되든지 아니면 정황에 따라 준비나 페색대기렬에 배치된다. 끝으로 사건이 발생하면 사건은 기다리고 있는 페색대기렬에 있는 모든 프로쎄스들의 준비대기렬에로 이동된다.

이 후자의 방법은 사건이 발생할 때 조작체계가 전체 페색된 대기렬을 주사하여 사건을 기다리는 프로쎄스들을 찾아 내야 한다는것을 의미한다. 대용량조작체계에서는 대기렬에 수백 지어 수천개의 프로쎄스가 있을수 있다. 따라서 매개 사건별로 하나씩 몇개의 대기렬을 가지는것이 보다 편리할것이다. 그러면 사건이 발생할 때 해당 대기렬에 있는 전체 프로쎄스들의 목록을 준비상태에로 이동시킬수 있다(그림 3-7 ㄴ).

마지막으로 강조해야 할 한가지 문제가 있다. 프로쎄스들의 할당을 우선권방안에

---

[5] 일반적으로 선취라는 용어는 어떤 프로쎄스로부터 프로쎄스가 그에 대한 사용을 끝마치기전에 어떤 자원을 반환할것을 요구하는것으로 정의되여 있다. 이 경우에 자원은 처리기자체이다. 프로쎄스는 집행중이며 계속 집행할수 있으나 선취하여 다른 프로쎄스를 집행할수 있다.

따라 시행한다면 매개 우선권준위에 하나씩 몇개의 준비대기렬을 가지는것이 편리하다. 그렇게 되면 조작체계는 어느것이 가장 높은 우선권을 가지고 가장 오래 기다린 프로쎄스인가를 쉽게 판단할수 있다.

중단된 프로쎄스

교체의 필요성

방금 설명한 세개의 원리적인 상태(준비, 실행, 페색)는 프로쎄스의 성질을 모형화하는 체계적인 방법을 주며 조작체계의 실현을 안내해 준다. 많은 조작체계들은 바로 이 세개의 상태들을 사용하여 구성된다.

그러나 그 모형에 다른 상태들을 추가하는 더 좋은 정식화가 있다. 이 새로운 상태들의 편리성을 보기 위해 가상기억기를 사용하지 않는 체계를 고찰하자. 집행할 매개 프로쎄스는 주기억기에 총체적으로 적재하여야 한다. 이렇게 하여 그림 3-7 ㄴ에서 모든 대기렬에 있는 프로쎄스전부를 주기억기에 상주시켜야 한다.

이제 이 정교한 기계장치모두에서의 전제는 입출력동작들이 계산작업보다 훨씬 느리므로 단일프로그람처리체계에서 처리기가 그 시간동안에 대부분 휴식하는데 있다는것을 상기해 보자. 그러나 그림 3-7 ㄴ의 방법은 이 문제를 완전히 해결하지 못한다. 이 경우에 기억기가 여러개의 프로쎄스들을 유지하고 있으며 한 프로쎄스가 기다리고 있을 때 다른 프로쎄스에로 처리기를 이동시킬수 있다는것은 사실이다. 그러나 처리기가 입출력보다 훨씬 더 빠르므로 입출력을 기다리고 있는것은 기억기의 모든 프로쎄스들에서 공통적일것이다. 이런데로부터 다중프로그람처리를 할 때에도 처리기가 그 시간에 대부분 휴식할수 있다.

무엇을 하겠는가? 더 많은 프로쎄스들을 병합시킬수 있게 주기억기를 확장할수도 있다. 그러나 이 방법에는 두가지 결함이 있다. 첫째로, 주기억기와 관련한 비용문제가 있는데 비트당으로 보면 작을수 있지만 메가비트, 기가비트의 기억기를 쓰는데 따라 비용이 늘어 난다. 둘째로, 기억기에 대한 프로그람들의 요구는 기억기의 비용이 떨어 지는것만큼 빨리 높아 진다. 그래서 기억기가 클수록 보다 많은 프로쎄스가 아니라 보다 큰 프로쎄스를 산생시키고 있다.

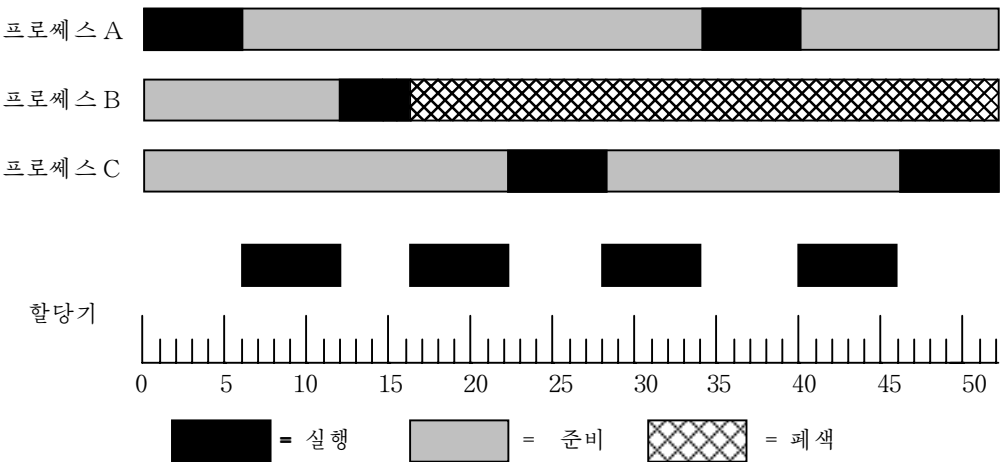



그림 3-6. 그림 3-3의 추적을 위한 프로쎄스의 상태

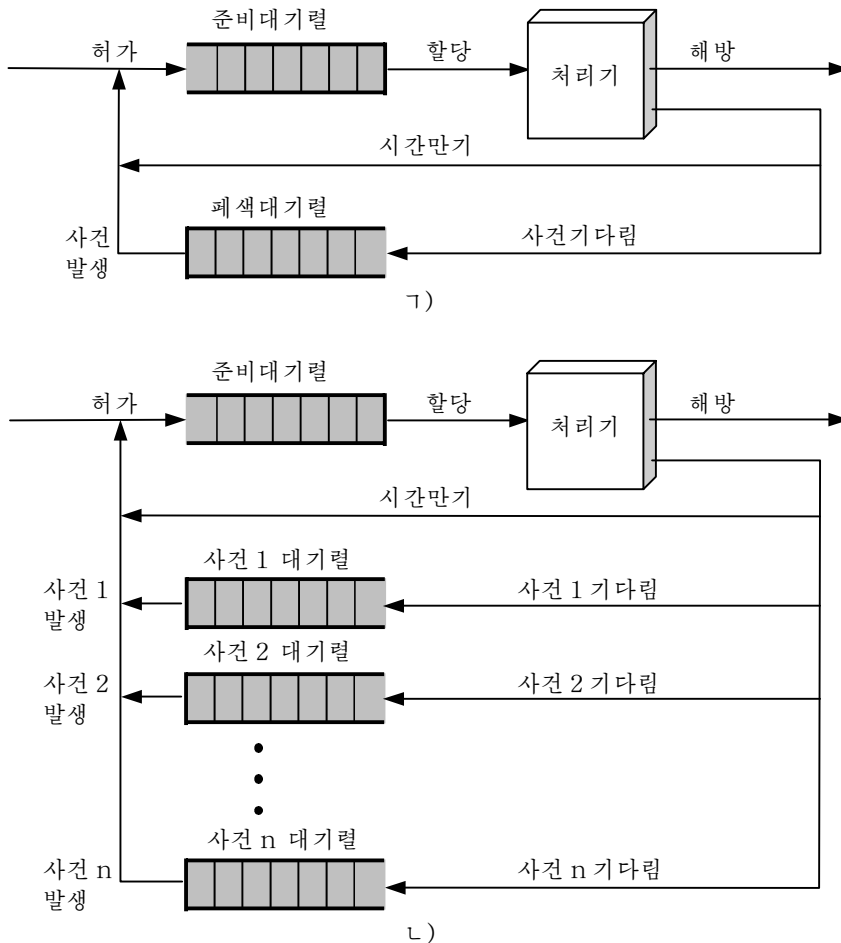



그림 3-7. 그림 3-5 에 대한 대기렬모형
ㄱ-단일페색대기렬, ㄴ-다중페색대기렬

다른 하나의 해결방도는 교체조작인데 이것은 프로쎄스의 일부 또는 전부를 주기억기에서 디스크에로 이동시키는것이다. 주기억기의 프로쎄스중에서 아무것도 준비상태에 있지 않을 때 조작체계는 페색된 프로쎄스들중의 한개를 디스크상의 중단대기렬로 교체시킨다. 이것은 주기억기에서 림시로 뽑아 버린 즉 중단시킨 현존프로쎄스의 대기렬이다. 조작체계는 후에 중단대기렬에서 다른 프로쎄스를 가져 오든가 새로운 프로쎄스요청에 권한을 준다. 다음 새롭게 도착한 프로쎄스에 대한 집행을 계속한다.

그러나 교체조작은 하나의 입출력조작이므로 문제를 더 나쁘게 만들어 버릴수 있다. 그렇지만 디스크입출력이 일반적으로 체계상에서 제일 빠른 입출력이므로(실례로 테프나 인쇄기입출력에 비하여) 교체조작은 일반적으로 성능을 높여 준다.

교체조작을 사용하자면 방금 설명한바와 같이 프로쎄스의 성질을 가지는 모형에 하나의 다른 상태 즉 중단상태를 추가해야 한다(그림 3-8 ㄱ). 주기억기의 모든 프로쎄스가 페색상태에 있다면 조작체계는 하나의 프로쎄스를 중단상태에 넣어 그것을 디스크에로 이송시켜 중단시킬수 있다. 주기억기에서 자유로워 진 공간은 다른 프로쎄스를 가져다 넣는데 사용할수 있다.

조작체계가 교체내기조작을 하였을 때 선택하는 프로쎄스를 주기억기에 가져다 넣는데는 두가지 경우가 있다. 즉 새롭게 창조된 프로쎄스를 수용할수 있고 아니면 이미 중단된 프로쎄스를 넣을수 있다. 체계상에 총적인 적재를 증가시키기보다는 봉사로서 그것을 보장하기 위해서 이미 중단된 프로쎄스를 가져다 넣는 편이 더 좋을것이다.

그러나 이 방향의 추리는 일정한 곤난을 조성한다. 중단된 모든 프로쎄스들은 중단할 때 페색상태에 있었다. 페색된 블로크를 주기억기에 다시 가져다 넣는것은 명백히 좋지 못하다. 왜냐하면 그것은 아직 집행할 준비가 되여 있지 않기때문이다. 그러나 중단상태에 있는 매개 프로쎄스는 원래 어떤 특정한 사건에 따라 페색되였다는것을 주의하자. 그러한 사건이 발생하면 프로쎄스는 페색되지 않으며 집행할수 있는 가능성을 가지게 된다.

따라서 이런 설계국면을 다시 생각해 볼 필요가 있다. 여기에 두개의 독립적인 개념이 있다. 즉 프로쎄스가 어떤 사건을 기다리는가 아닌가 하는것(페색 또는 비페색)과 프로쎄스가 주기억기밖으로 교체되였는가 안되였는가(중단 또는 비중단)하는 개념들이다. 이 2*2조합을 적응시키자면 4개의 상태가 필요하다. 즉

- **준비** : 프로쎄스가 주기억기에 있으며 실행할수 있다.
- **페색** : 프로쎄스가 주기억기에 있으며 어떤 사건을 기다리고 있다.
- **페색/중단** : 프로쎄스가 2차기억기에 있으며 어떤 사건을 기다리고 있다.
- **준비/중단** : 프로쎄스가 2차기억기에 있지만 주기억기에 적재되자마자 집행할수 있다.

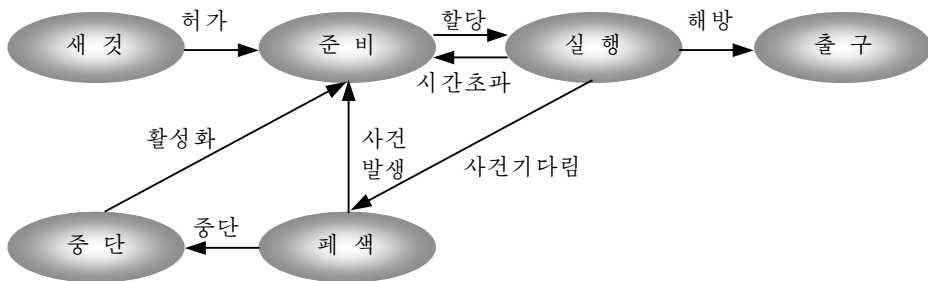
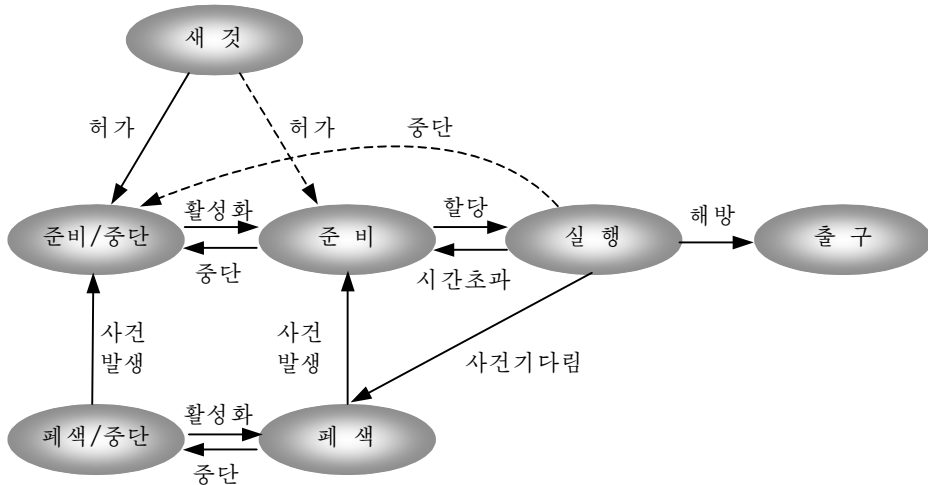



그림 3-8. 중단상태를 가지는 프로쎄스의 상태이행도
ㄱ-중단상태가 한개인 경우, ㄴ-중단상태가 두개인 경우

두개의 새로운 중단상태를 포함하고 있는 상태이행도를 보기전에 한가지 다른 론점을 언급하기로 한다. 지금까지 론의에서는 가상기억기를 사용하지 않으며 프로쎄스가 전부 주기억기에 있거나 주기억기밖에 있다고 가정하였다. 가상기억기방안을 사용하면 부분적으로만 주기억기에 있는 프로쎄스를 집행할수 있다. 주기억기에 없는 프로쎄스주소를 참조하게 된다면 프로쎄스의 해당 부분을 가져다 넣을수 있다. 가상기억기를 사용하면 명백한 교체에 대한 요구가 높아 지게 된다. 왜냐하면 임의의 희망하는 프로쎄스내에서 임의의 희망하는 주소를 처리기의 기억기관리하드웨어에 의해 주기억기의 안팎으로 이동시킬수 있기때문이다. 그러나 제8장에서 보게 되는바와 같이 프로쎄스의 수가 충분히 많고 그 모두가 부분적으로 주기억기에 있다면 가상기억기체계의 성능은 떨어 질수 있다. 따라서 가상기억기체계라고 해도 조작체계는 성능을 위하여 프로쎄스들을 시간에 따라 명백하고 완전하게 교체해야 한다.

이제 앞에서 제기한 상태이행모형에 있는 그림 3-8 ㄴ를 보자(그림에서 점선으로 련결된 선들은 가능하나 이행할 필요가 없다는것을 가리키고 있다.). 새로운 의의 있는 이행들은 다음과 같다.

- **페색→페색/중단 :** 준비상태에 있는 프로쎄스가 전혀 없다면 적어도 하나의 페색된 프로쎄스를 교체하여 페색되지 않은 다른 프로쎄스에 빈 자리를 내주어야 한다. 집행할수 있는 준비상태의 프로쎄스들이 있다고 하여도 조작체계가 현재 실행하고 있는 프로쎄스나 할당하려고 하는 준비상태의 프로쎄스가 충분한 성능을 유지하기 위해 주기억기를 더 요구한다는것을 판단하면 이러한 이행이 일어 날수 있다.
- **페색/중단→준비/중단 :** 기다리고 있던 사건이 발생하면 페색/중단상태에 있는 프로쎄스를 준비/중단상태에로 이동시킨다. 이것은 중단된 프로쎄스와 관련된 상태정보가 틀림없이 조작체계에 접근할수 있어야 한다는 요구를 제기하고 있다는데 주목하자.
- **준비/중단→준비 :** 주기억기에 준비된 프로쎄스가 전혀 없다면 조작체계는 하나를 끌어다 넣고 집행을 계속해 나가야 한다. 게다가 준비/중단상태에 있는 프로쎄스가 준비상태에 있는 임의의 프로쎄스보다 더 높은 우선권을 가지는 경우가 있을수 있다. 그 경우에 조작체계설계자는 보다 높은 우선권을 가진 프로쎄스를 끌어들이는것이 교체를 최소화하는것보다 더 중요하다고 볼수 있다.
- **준비→준비/중단 :** 보통 조작체계는 준비된 프로쎄스보다 페색된 프로쎄스를 중단시키는쪽을 택하려고 한다. 그것은 준비된 프로쎄스는 이제 집행시킬수 있지만 반면에 페색된 프로쎄스는 주기억공간을 많이 차지하면서 집행시킬수 없기때문이다. 그러나 충분히 큰 주기억블로크를 자유롭게 할 방도로 되는 경우에 준비된 프로쎄스를 중단시켜야 한다. 또한 조작체계는 페색된 프로쎄스가 곧 준비된 프로쎄스로 되리라는 믿음이 있는 경우 높은 우선권을 가진 페색된 프로쎄스가 아니라 낮은 우선권을 가진 준비된 프로쎄스를 선택하여 중단시킬수 있다.

고려할 가치가 있는 몇가지 다른 이행들에 대하여 보자.

- **새것→준비/중단과 새것→준비 :** 새로운 프로쎄스가 창조될 때 그것을 준비대기렬에 추가하거나 준비/중단대기렬에 추가할수 있다. 어느 경우에나 조작체계는 일정한 표를 만들어 프로쎄스를 관리하고 프로쎄스에 주소공간을 배정하여야 한다.

조작체계가 초시기에 이 보조조작을 수행하는것이 좋을수 있다. 그렇게 하여야 폐색되지 않는 큰 프로쎄스집결소를 유지할수 있다. 이 전략에 따르면 흔히 새로운 프로쎄스들에 대해 주기억기의 공간이 불충분해 지게 되는데 이로부터 새것→준비/중단이행을 사용하게 된다. 다른 한편 최근에 가능한 프로쎄스를 창조하는 최근시간원리는 조작체계의 간접소비시간을 감소시키고 폐색된 프로쎄스들로 체계가 충만되는 시각에 조작체계가 프로쎄스를 창조할수 있게 해준다고 볼수 있다.

- **폐색/중단→폐색** : 이런 이행을 포함시키는것은 서툰 설계인것처럼 보일수 있다. 결국 프로쎄스가 실행할 준비가 되여 있지 않으며 이미 주기억기에 없다면 그것을 끌어 들이는 요점은 무엇인가? 그러나 다음과 같은 각본을 고찰하여 보자. 프로쎄스가 완료하고 일정한 주기억기를 자유롭게 한다. 폐색/중단대기렬에 프로쎄스가 있는데 준비/중단대기렬의 임의의 프로쎄스보다 더 높은 우선권을 가지고 있으며 조작체계는 그 프로쎄스를 폐색시키는 사건이 곧 발생한다는것을 확인시켜 주는 리유를 알고 있다. 이런 상황에서 폐색된 프로쎄스를 주기억기에 가져오는것이 준비된 프로쎄스를 가져 오는것보다 합리적이라고 볼수 있다.
- **실행→준비/중단** : 보통 실행중인 프로쎄스는 할당된 시간이 끝나면 준비상태로 이동된다. 그러나 폐색/중단대기렬상의 보다 높은 우선권을 가진 프로쎄스가 방금 폐색에서 벗어 나게 되였기때문에 그 프로쎄스를 선취하려고 한다면 조작체계는 실행중인 프로그람을 즉시에 준비/중단대기렬에 이동시키고 일부 주기억기를 자유롭게 할수도 있다.
- **각이한 상태→출구** : 대체로 프로쎄스는 실행하는 동안에 완료한다. 그것은 프로쎄스가 완료되였거나 일부 치명적인 오유조건때문이다. 그러나 일부 조작체계들에서 프로쎄스는 그것을 창조한 프로쎄스에 의해 또는 부모프로쎄스자체가 완료할때 완료될수 있다. 이것을 허용한다면 임의의 상태에 있는 프로쎄스를 출구상태에로 이동시킬수 있다.

중단의 다른 사용

지금까지는 중단된 프로쎄스에 대한 개념을 주기억기에 있지 않는 프로쎄스에 대한 개념과 동등하게 보았다. 주기억기에 있지 않는 프로쎄스는 그것이 어떤 사건을 기다리든 기다리지 않든 즉시에 집행에 인입할수 없다.

여기서는 중단된 프로쎄스에 대한 개념을 일반화할수 있다. 중단된 프로쎄스를 다음과 같은 특성을 가지는것으로 정의하자. 즉

1. 프로쎄스는 즉시에 집행에 인입할수 없다.
2. 프로쎄스는 어떤 사건을 기다릴수도 있고 기다리지 않을수도 있다. 그렇다면 이 폐색조건은 중단조건에 무관계하며 폐색사건의 병행은 프로쎄스를 집행할수 없게 한다.
3. 프로쎄스 그자체든지 부모프로쎄스든지 또는 그의 집행을 막기 위한 목적으로 조작체계가 일으킨 어떤 원인에 의하여 프로쎄스가 중단상태에 놓이게 되였다.
4. 명백하게 이동을 지시할 때까지 프로쎄스를 이 상태에서 재이동시킬수 없다.

표 3-3은 프로쎄스의 중단에 대한 몇가지 리유를 서술하고 있다. 설명한 한가지 리유는 준비된 프로쎄스를 끌어 들이거나 다른 말로 가상기억기체계에 대한 부담도 덜어주어 남아 있는 매개 프로쎄스가 더 많은 주기억기를 가지도록 프로쎄스를 디스크에로

교체하는것이다. 조작체계가 프로쎄스를 중단시키는데서 다른 필요성이 있을수도 있다. 실례로 검열 또는 추적프로쎄스를 체계의 동작을 감시하는데 사용할수 있다. 즉 프로쎄스를 여러가지 자원들(처리기, 기억기, 통로들)의 사용준위와 체계에서 사용자프로쎄스의 진행률을 기록하는데 사용할수 있다. 조작자의 조종하에서 조작체계는 이런 프로쎄스를 시간에 따라 투입, 차단시킬수 있다. 만일 조작체계가 문제를 발견하거나 짐작한다면 프로쎄스를 중단시킬수 있다. 이에 대한 한가지 실례가 교착이다. 이것은 제6장에서 설명한다. 다른 실례로 통신회선상에서 문제를 발견하면 조작자는 조작체계를 가지고 일정한 시험을 실행하는 동안 그 회선을 사용하는 프로쎄스를 중단시킨다.

표 3-3. 프로쎄스중단의 리유

교체	조작체계는 집행할 준비가 되여 있는 프로쎄스를 끌어 들이기 위하여 충분한 주기억기를 해방시켜야 한다.
다른 조작체계	조작체계는 배경 즉 편의프로쎄스나 문제가 있는 프로쎄스를 중단시킬수 있다.
대화형사용자요청	사용자가 착오수정을 목적으로 또는 자원의 사용과 관련하여 프로그람 집행을 중단시키려고 할수 있다.
시간맞추기	프로쎄스는 주기적으로 집행될수 있으며(실례로 회계 또는 체계감시프로쎄스) 다음시간간격을 기다리는동안 중단될수 있다.
부모프로쎄스의 요청	부모프로쎄스가 중단된 프로쎄스를 조사하거나 수정하기 위해서 또는 여러 자식들의 동작을 일치시키기 위하여 자식프로쎄스의 집행을 중단시키려고 할수 있다.

다른 리유들은 대화형사용자의 동작과 관련되여 있다. 실례로 사용자가 프로그람에서 오유를 발견하면 누구든 그 집행을 중단하고 프로그람이나 자료를 검사 및 변경하여 착오를 수정하고 집행을 회복할수 있다. 또한 사용자가 투입, 차단시킬수 있는 추적이나 관심하는 통계적값들을 모아 놓고 있는 배경프로쎄스가 있을수 있다.

시간맞추기도 교체를 결심하게 할수 있다. 실례로 프로쎄스가 주기적으로 활성화되지만 시간의 대부분을 휴식한다면 사용들사이에서 교체된다. 사용률 또는 사용자활성도를 감시하는 프로그람이 하나의 실례로 된다.

끝으로 부모프로쎄스가 자식프로쎄스들을 중단시키려고 할수 있다. 실례로 프로쎄스 A가 프로쎄스 B를 새끼 쳐서 파일읽기를 할수 있다. 그다음에 프로쎄스 B가 파일읽기수속에서 어떤 오유와 마주 쳐서 이것을 프로쎄스 A에 보고한다. 프로쎄스 A는 프로쎄스 B를 중단시키고 그 원인을 조사한다.

이 모든 경우에 중단된 프로쎄스의 활성화는 처음에 중단을 요청한 대행자가 요청한다.

제 2 절. 프로쎄스의 서술

조작체계는 콤퓨터체계에서 사건들을 조종한다. 그것은 처리기가 집행하도록 일정작성을 하고 프로쎄스들을 할당하며 자원들을 프로쎄스들에 배정하여 기본봉사를 받으려는 사용자프로그람들의 요청에 응답한다. 기본적으로 조작체계를 프로쎄스들이 체계자원을 사용하는것을 관리하는 개체라고 생각할수 있다.

이 개념은 그림 3-9에서 설명하고 있다. 다중프로그람처리환경에는 창조되여 가상기억기에 존재하는 몇개의 프로쎄스(P1, . . . , Pn)가 있다. 매개 프로쎄스는 집행과정에 처

리기입출력장치들과 주기억기를 포함하는 일정한 체계자원에 접근할것을 요구한다. 그림에서 프로쎄스 P1은 실행하고 있다. 즉 적어도 프로쎄스의 일부가 주기억기에 있고 두개의 입출력장치에 대한 조종을 가지고 있다. 프로쎄스 P2는 역시 주기억기에 있지만 P1에 배정된 입출력장치를 기다리면서 페색되여 있다. 프로쎄스 Pn은 교체되여 나갔고 따라서 중단되여 있다.

뒤의 장들에서 프로쎄스들을 대신하여 조작체계가 이 자원을 관리하는 세부들을 고찰한다. 여기서는 더 기본적인 문제들에 관심을 둔다. 즉 무슨 정보가 프로쎄스들을 조종하며 그것들을 위한 자원을 관리하도록 조작체계에 요구하는가 하는것이다.

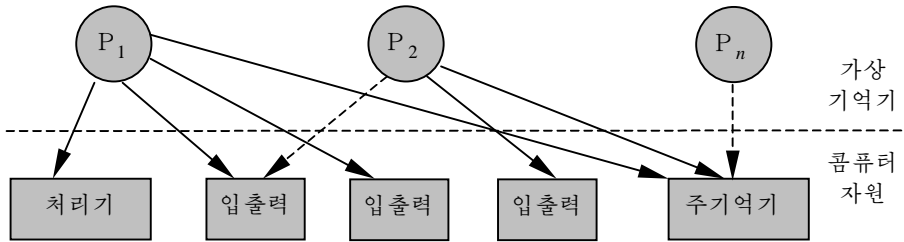



그림 3-9. 프로쎄스와 자원(당시 하나의 순시상태에서의 자원배정)

조작체계의 조종구조

조작체계가 프로쎄스와 자원들을 관리한다면 분명 매개 프로쎄스와 자원의 현재 상태에 대한 정보를 가지고 있어야 한다. 이 정보를 주는 만능적인 방법은 간단하다. 즉 관리하게 될 매개 객체에 대하여 조작체계가 정보표를 구성하고 유지하는것이다. 이 조작에 대한 일반적표상을 그림 3-10에서 지적하고 있는데 이것은 조작체계가 기억기, 입출력, 파일과 프로쎄스의 4가지 서로 다른 형의 표들을 유지하고 있다는것을 보여 주고 있다. 세부들은 조작체계마다 다를수 있지만 모든 조작체계들은 기본적으로 이 4개의 범주에 속하는 정보를 유지한다.

기억기표는 주(실) 및 2차(가상)기억기모두에 대한 추적을 보존하는데 쓰인다. 주기억기의 일부는 조작체계가 사용하도록 예약되여 있고 나머지는 프로쎄스들이 사용할수 있다. 프로쎄스들은 일정한 종류의 가상기억기나 단순한 교체수법을 사용하여 2차기억기상에 유지되여 있다. 기억기표는 다음과 같은 정보를 유지하고 있어야 한다. 즉

- 프로쎄스들에 대한 주기억기의 배정
- 프로쎄스들에 대한 2차기억기의 배정
- 어느 프로쎄스가 일정한 공유기억구역에 접근할수 있는가 하는것과 같은 주 또는 가상기억기블로크들에 대한 보호속성
- 가상기억기를 관리하는데 필요한 정보.

제3편에서는 가상기억기관리에서의 정보구조를 세부적으로 보게 된다.

입출력표는 조작체계가 콤퓨터체계의 입출력장치와 통로들을 관리하는데 쓰인다. 임의로 주어 진 시간에 입출력장치를 차지하거나 어떤 특정한 프로쎄스에 할당할수 있다. 입출력조작이 진행중이면 조작체계는 입출력조작의 상태와 입출력이송의 원천지와 목적지로 사용되는 주기억기의 위치를 알아야 한다. 입출력관리는 제11장에서 고찰한다.

조작체계는 또한 **파일표**를 유지할수 있다. 이 표들은 파일의 존재성, 2차기억기상에서 그것들의 위치, 그것들의 현 상태와 다른 속성들에 대한 정보를 준다. 이 정보의 전부는 아니지만 많은 몫을 파일관리체계가 유지하고 사용할수 있는데 이 경우에 조작체계는 파일에 대한 지식이 적거나 전혀 없다. 다른 조작체계들에서 파일관리의 많은 세부들은 조작체계자체가 관리한다. 이 문제는 제12장에서 고찰한다.

끝으로 조작체계는 프로쎄스들을 관리하기 위하여 프로쎄스표를 유지하고 있어야 한다. 이 절의 나머지 부분에서는 요구되는 **프로쎄스표**를 고찰한다. 이 설명에 들어 가기전에 앞서 두가지 문제를 고찰해야 한다. 첫째로, 그림 3-10에서 4개의 명백한 표들을 보여 주고 있는데 이 표들이 일정한 방식으로 련결되거나 호상 참조되여야 한다는것은 명백하다. 프로쎄스들을 대신하여 기억기, 입출력과 파일들을 관리하며 그래야 프로쎄스의 표들에서 직접적으로 또는 간접적으로 자원들을 일정하게 참조할수 있다. 파일표로 참조되는 파일들은 입출력장치를 경유하여 접근할수 있으며 그 파일들은 주기억기나 가상기억기에 있게 된다. 표들자체는 조작체계가 접근할수 있어야 하며 따라서 기억기관리하에 있게 된다.

둘째로, 조작체계가 처음에 표들을 창조하는것을 어떻게 알겠는가? 명백히 조작체계는 기억기가 얼마나 많은가, 어떤 입출력장치들이 있는가, 그것들의 식별자는 무엇인가 등과 같은 기초적환경에 대한 일정한 지식을 가지고 있어야 한다. 이것은 구성상 문제이다. 즉 조작체계가 초기화될 때 기초적환경을 정의하는 일정한 구성자료에 접근해야 하며 이 자료들은 인간의 방조나 일정한 자동구성쏘프트웨어에 의해 조작체계밖에서 창조되여야 한다.

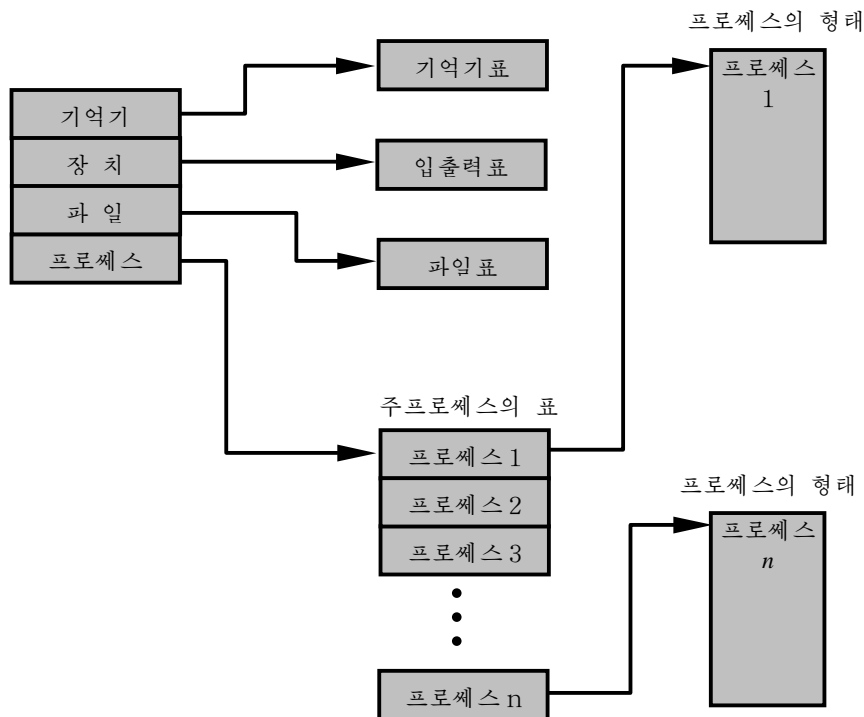



그림 3-10. 조작체계조종표의 일반적구조

프로쎄스의 조종구조

프로쎄스를 관리하고 조종할 때 조작체계가 무엇을 알아야 하는가에 대하여 고찰하자. 첫째로, 프로쎄스가 어디에 있는가를 알아야 하며 둘째로, 그것을 관리하는데 필요한 조작체계의 속성들을 알고 있어야 한다(실례로 프로쎄스 ID, 프로쎄스의 상태, 기억기에서의 위치).

프로쎄스의 위치

프로쎄스가 어디에 있는가 또는 그의 속성이 무엇인가에 대한 문제를 취급하자면 몇가지 기본적인 문제 즉 프로쎄스의 물리적인 명시는 무엇인가 하는것을 더 이야기해야 한다. 최소한 프로쎄스는 집행되여야 할 프로그람이나 프로그람의 모임을 포함하고 있어야 한다. 이 프로그람과 관련된것들은 국부적 및 전역변수들에 대한 자료의 위치모임이며 어떤 정의된 상수들이다. 이렇게 함으로써 프로쎄스는 프로쎄스의 프로그람과 자료를 유지하는데 최소로 충분한 기억기로 구성된다. 게다가 프로그람의 집행은 대체로 수속호출의 추적과 수속들사이에서 넘겨 주는 파라메터들을 유지하는데 사용되는 탄창을 포함한다(부록 1-ㄱ를 보시오.). 결국 매개 프로쎄스는 프로쎄스의 조종을 위해 조작체계가 사용하는 몇개의 속성들과 관계를 가지고 있다. 대체로 속성들의 집합을 **프로쎄스조종블로크**[6]라고 한다. 여기서는 프로그람, 자료, 탄창 및 속성들의 집합을 **프로쎄스형태**(표 3-4)라고 할수 있다.

프로쎄스형태의 위치는 사용하는 기억기관리방안에 의존하게 될것이다. 간단한 경우에 프로쎄스형태는 잇닿은 즉 련속적인 기억기블로크로 유지된다. 이 블로크는 2차기억기 보통 디스크에 유지된다. 그래서 조작체계는 프로쎄스를 관리할수 있으며 적어도 그 형태의 작은 부분은 주기억기에 유지되여 있어야 한다. 이리하여 조작체계는 디스크에서 매개 프로쎄스의 위치와 주기억기에 있는 매개 프로쎄스에 대해 주기억기에서의 프로쎄스의 위치를 알아야 한다. 제2장에서는 CTSS조작체계에 대해 이 방안에서의 약간 더 복잡한 변동을 보았다. CTSS를 본다면 프로쎄스가 교체되여 나갈 때 프로쎄스형태의 일부는 주기억기에 남아 있을수 있다. 이런데로부터 조작체계는 매개 프로쎄스의 어느 부분의 형태에 대한 추적이 아직 주기억기에 있는가 하는것을 보존하고 있어야 한다.

대부분의 현대조작체계들은 기억기관리방안을 사용하는데 거기서 프로쎄스형태는 잇닿아 기억시킬것을 요구하지 않는 블로크들의 모임으로 구성되여 있다. 사용하는 방안에 따라 블로크들은 가변길이(토막이라고 부른다.) 또는 고정길이(폐지라고 부른다.) 또는 그의 결합으로 될수 있다. 임의의 경우에 그러한 방안은 조작체계로 하여금 임의의 특정한 프로쎄스의 일부분만을 가져다 넣게 한다. 이리하여 임의의 주어 진 시간에 부분적인 프로쎄스형태가 주기억기에 있을수 있으며 나머지는 2차기억기에 있게 된다.[7] 따라서 조작체계가 유지하는 프로쎄스표들은 매개 프로쎄스형태의 매개 토막이나 매개 폐지에 대한 위치를 표시하여야 한다.

---

6. 이 자료구조에 대하여 일반적으로 쓰이는 다른 이름은 과제조종블로크, 프로쎄스서술자 및 과제서술자이다.

7 이 간단한 설명은 좀 세부에 치우친다. 특히 가상기억기를 사용하는 체계에서 능동프로쎄스에 대한 모든 프로쎄스형태는 항상 2 차기억기에 있다. 그 형태의 일부를 주기억기에 적재할 때 그것은 이동시킨다기보다 복사된다. 이렇게 하여 2차기억기는 모든 토막이나 모든 폐지의 복사물을 기억한다. 그러나 그 형태의 주기억기부분이 변경된다면 주기억기부분을 디스크에 복사하기전까지는 2 차복사가 갱신되지 않는 상태에 있게 된다.

표 3-4. 프로쎄스형태의 대표적요소

사용자자료 사용자공간의 변경할수 있는 부분. 프로그람자료, 사용자의 탄창구역 및 변경될수 있는 프로그람들을 포함할수 있다.
사용자프로그람 집행될 프로그람
체계탄창 매개 프로쎄스는 그와 관련된 한개이상의 후입선출체계탄창을 가지고 있다. 탄창은 파라메터들과 수속 및 체계호출에서의 호출주소를 보관하는데 쓰인다.
프로쎄스조종블로크 조작체계가 프로쎄스를 조종하기 위해 필요로 하는 자료(표 3-6을 보시오.)

그림 3-10은 위치정보의 구조를 다음과 같은 방법으로 묘사하고 있다. 매개 프로쎄스에 대해 한개 입구점을 가진 초기프로쎄스표가 있다. 매개 입구점은 적어도 프로쎄스형태에 대한 지시기를 포함하고 있다. 프로쎄스형태가 여러개의 블로크를 포함하고 있다면 이 정보는 직접 초기프로쎄스표에 포함되거나 기억기표의 입구점들에 대해 호상 참조할수 있다. 물론 이 묘사는 일반적이다. 즉 특정한 조작체계는 그자체의 위치정보조직방법을 가질수 있다.

프로쎄스의 속성

복잡하게 얽힌 다중프로그람식체계는 매개 프로쎄스에 대한 많은 정보를 요구한다. 설명된바와 같이 이 정보는 프로쎄스조종블로크에 머물러 있는것으로 고찰할수 있다. 각이한 체계들은 이 정보를 서로 다른 방법으로 조직하게 되는데 이에 대한 몇가지 실례를 이 장의 마지막과 다음장에서 찾아 볼수 있다. 현재로서는 그 정보를 어떻게 조직하는가 하는 구체적인 세부를 고려하지 않고 조작체계들에서 사용할수 있는 정보의 형태를 간단히 고찰해 보자.

표 3-5는 매개 프로쎄스에 대해 조작체계가 요구하는 정보의 대표적인 범주들을 목록화한것이다. 요구되는 정보의 량을 보면 어느 정도 놀랄수도 있다. 조작체계의 부담에 대해 더 크게 인정할 때 이 목록은 보다 더 합리적인것으로 될것이다.

프로쎄스의 조종블로크정보를 세가지 일반적범주들로 그룹화할수 있다. 즉

- 프로쎄스의 식별
- 처리기의 상태정보
- 프로쎄스의 조종정보

프로쎄스식별에 대하여 말할 때 실제상 모든 조작체계들에서 매개 프로쎄스에는 단일한 수자식별자호가 붙게 되며 그것은 단순히 본래 프로쎄스표에 들어 가는 첨수일수 있다(그림 3-10). 그렇지 않으면 조작체계가 프로쎄스식별자에 기초하여 적당한 표를 찾을수 있는 배치로 되여야 한다. 이 식별자를 몇가지 방법으로 사용할수 있다. 조작체계가 조종하는 많은 다른 표들이 프로쎄스표들을 호상 참조하기 위하여 프로쎄스식별자를 사용할수 있다. 실례로 기억기표를 조직하여 어느 프로쎄스가 어느 구역에 설정되는가를 지적해 주는 주기억기배치를 줄수 있다. 류사한 참조가 입출력 및 파일표들에서 있을수 있다. 프로쎄스들이 서로 통신할 때 프로쎄스식별자는 특정한 통신목적지를 조작체계에 통지한다. 프로쎄스들이 다른 프로쎄스들을 창조할 허락을 받게 될 때 식별자는 매개 프로쎄스에 대해 부모와 자식을 가리켜 준다.

표 3-5. 프로쎄스조종블로크의 대표적요소들

프로쎄스의 식별

식별자

프로쎄스조종블로크와 함께 기억되여야 할 수자식별자에 포함되는것

- 이 프로쎄스의 식별자
- 이 프로쎄스를 창조하는 프로쎄스(부모프로쎄스)의 식별자
- 사용자의 식별자

처리기의 상태정보

사용자변경등록기

사용자변경등록기는 처리기가 진행하는 기계어를 사용하여 참조할수 있는 등록기이다. 대체로 이런 등록기들이 8～32개 있으며 일부 RISC실현물들은 100개이상을 가지고 있다.

조종 및 상태등록기

여러가지 처리기의 등록기들이 있는데 이것은 처리기의 조작을 조종하는데 쓰인다. 포함되는 내용은

- **프로그람계수기** : 불러 내야 할 다음명령의 주소를 담고 있다.
- **조건코드** : 가장 최고의 산수 또는 론리연산의 결과(실례는 부호, 령, 자리올림, 같기, 자리넘침)
- **상태정보**:새치기의 가능/불가능기발, 집행방식을 포함한다.

탄창

매개 프로쎄스는 그와 관련된 한개이상의 후입선출체계탄창을 가지고 있다.
탄창은 수속 및 체계호출에서의 파라메터들과 호출주소를 기억시키는데 사용한다

프로쎄스의 조종정보

일정작성 및 상태정보

이것은 일정작성을 수행하기 위하여 조작체계에 요구되는 정보이다. 정보의 대표적인 항목들은 다음과 같은것들을 포함하고 있다.

- **프로쎄스상태**:집행을 위해 일정작성하게 될 프로쎄스의 준비정도를 정의한다(실례로 실행, 준비, 기다림, 정지)
- **우선권** : 프로쎄스의 일정작성우선권을 서술하기 위해 한개이상의 마당을 사용할수 있다. 일부 체계들에서는 몇가지 값들을 요구한다(실례로 초기값, 현재우선권, 허용할수 있는 가장 높은 우선권)
- **일정작성관련정보** : 이것은 사용하는 일정작성알고리듬에 의존한다. 실례는 프로쎄스가 기다린 시간의 량과 프로쎄스가 마지막시기에 집행한 시간의 량을 들수 있다.
- **사건** : 프로쎄스가 회복될 가능성을 가지기전에 기다리는 사건의 정체

자료의 구조화

프로쎄스를 대기렬, 고리, 일부 다른 구조에서 다른 프로쎄스와 련결할수 있다. 실례로 특정한 우선권준위에서 기다림상태에 있는 모든 프로쎄스를 하나의 대기렬에서 련결시킬수 있다. 이런 프로쎄스는 다른 프로쎄스와 부모-자식(창조자-피창조자)관계를 나타낼수 있다. 프로쎄스조종블로크는 이 구조를 지원하는 다른 프로쎄스들에 대한 지시기들을 포함할수 있다.

프로쎄스사이 통신

여러가지 기발, 신호 및 통신들이 두개의 독립적인 프로쎄스들사이에서의 통신과 관계될수 있다. 이 정보의 일부 또는 전부를 프로쎄스의 조종블로크에서 유지할수 있다.

표계속

프로쎄스의 특권 프로쎄스들은 접근할수 있는 기억기와 집행할수 있는 명령의 행에 의하여 특권을 허가 받는다. 더우기 체계편의프로그람과 봉사를 사용하는데 특권을 적용할수 있다.
기억기관리 이 프로쎄스에 할당된 가상기억기를 묘사하는 토막 또는 페지표에 대한 지시기를 포함할수 있다.
자원소유권과 사용률 열린 파일과 같은 프로쎄스가 조종하는 자원을 가리킬수 있다. 처리기나 다른 자원의 사용률경력도 포함할수 있다. 즉 일정작성프로그람이 이 정보를 요구할수도 있다.

이 프로쎄스식별자들외에 일감을 사용자가 책임진다는것을 가리키는 사용자식별자를 프로쎄스에 설정해 줄수 있다.

처리기의 상태정보는 처리기등록기들의 문맥으로 구성된다. 물론 프로쎄스가 실행하고 있는 동안 정보는 등록기들에 있다. 프로쎄스가 새치기될 때 이 모든 등록기정보는 프로쎄스가 집행을 회복할 때 불러 낼수 있도록 보관되여야 한다. 포함되는 등록기들의 특성과 수는 처리기의 설계에 따른다. 대체로 등록기조는 사용자변경등록기, 조종 및 상태등록기들 및 탄창들을 포함한다. 이것들은 제1장에서 설명하였다.

특히 주목할수 있는것은 모든 처리기의 설계는 흔히 프로그람상태단어(PSW)라고 하는 등록기나 등록기모임을 포함하는데 이것은 상태정보를 담고 있다. PSW는 대체로 조건코드들과 그밖에 다른 상태의 정보를 담고 있다. 처리기상태단어의 좋은 실례로 펜티움기계들을 들수 있는데 그것을 EFLAGS등록기라고 한다(그림 3-11과 표 3-6에서 보여 준다.). 펜티움콤퓨터상에서 실행하는 임의의 조작체계(UNIX와 Windows NT를 포함하여)들이 이 구조를 사용한다.

프로쎄스의 조종블로크에서 정보에 대한 세번째 주요범주에 더 좋은 이름을 붙인다고 하면 **프로쎄스조종정보**라고 할수 있다. 이것은 조작체계가 여러가지 능동프로쎄스들을 조종 및 조정하는데 필요한 추가적인 정보이다. 표 3-5의 마지막부분은 이 정보에 대한 범위를 지적하고 있다. 뒤의 장들에서 조작체계의 기능성에 대한 세부를 고찰하느라면 이 목록상에 있는 여러가지 항목들에 대한 필요성이 명백해 질것이다.

31 | 21 | 16 | 15 | 0

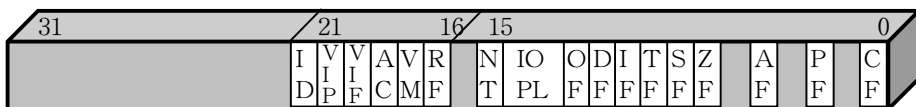
ID VIP VIF AC VM RF | NT IOPL OF DF IF TF SF ZF AF PF CF

ID	= 식별기발	DF	= 방향기발
VIP	= 가상새치기대기	IF	= 새치기가능기발
VIF	= 가상새치기기발	TF	= 함정기발
AC	= 정렬검사	SF	= 부호기발
VM	= 가상 8086 방식	ZF	= 령기발
RF	= 회복기발	AF	= 보조자리올림기발
NT	= 겹싼과제기발	PF	= 기우성기발
IOPL	= 입출력권한준위	CF	= 자리올림기발
OF	= 자리넘침기발		

그림 3-11. 펜티움 II EFLAGS 등록기

표 3-6. 펜티움의 EFLAGS등록기비트

조종비트	조작방식비트
AC(정렬검사) 어떤 단어나 배단어가 비단어나 비배단어 경계상에서 주소지정되면 설정한다. **ID(식별기발)** 이 비트를 설정 및 지우기할수 있으면 처리기는 CPUID명령을 지원한다. 이 명령은 판매자, 계렬, 모형에 대한 정보를 준다. **RF(회복기발)** 프로그람작성자에게 착오수정례외를 불가능하게 함으로써 착오수정례외후 즉시에 또다른 착오수정례외를 일으키지 않고 명령을 재출발시킬수 있다. **IOPL(입출력권한준위)** 설정했을때 보호방식조작기간 입출력장치들에 대한 모든 접근에 대해 처리기가 례외를 발생하게 한다. **DF(방향기발)** 문자렬처리명령들이 16bit반등록기 SI와 DI(16bit연산에서) 또는 32bit등록기 ESI와 EDI(32bit연산에서)를 증가시키겠는가 감소시키겠는가를 결정한다. **IF(새치기가능기발)** 설정했을 때 처리기는 외부새치기를 인식하게 된다. **TF(함정기발)** 설정했을 때 매개 명령의 집행후에 새치기를 일으킨다. 이것은 오유수정에 사용된다.	**NT(겹싼과제기발)** 현재의 과제가 보호방식조작에 있는 또다른 과제에 중복된다는것을 가리킨다. **VM(가상8086방식)** 프로그람작성자가 가상 8086방식을 가능 또는 불가능으로 할수 있게 하며 이것은 처리기가 8086기계어로 실행하는가 안하는가를 판정해 준다. **VIP(가상새치기기다림)** 가상8086방식에서 사용하여 한개이상의 새치기가 봉사를 기다리고 있다는것을 가리킨다. **VIF(가상새치기기발)** 가상 8086방식에서 IF대신 사용한다 **조건코드** **AF(보조자리올림기발)** AL등록기를 사용하는 8bit산수론리연산의 반바이트사이에서의 자리올림이나 꿔림을 표현한다. **CF(자리올림기발)** 산수연산명령후에 제일 높은 자리에 자리올림하거나 꿔 준다는것을 가리킨다. 일부 자리밀기 및 순환조작에 의해서도 수정된다. **OF(자리넘침기발)** 더하거나 덜기후에 연산자리초과가 일어났다는것을 가리킨다. **PF(기우성기발)** 산수 또는 론리연산결과의 기우성으로서 1은 우수성을 가리키며 0은 기수성을 가리킨다. **SF(부호기발)** 산수 또는 론리연산결과의 부호를 가리킨다. **ZF(령기발)** 산수 또는 론리연산결과가 0이라는것을 가리킨다.

그림 3-12는 가상기억기에서 프로쎄스형태에 대한 구조를 제기하고 있다. 매개 프로쎄스형태는 프로쎄스조종블로크, 사용자탄창, 프로쎄스의 전용주소공간 및 프로쎄스가 다른 프로쎄스와 공유하는 임의의 다른 주소공간으로 구성된다. 그림에서 매개 프로쎄스형태는 잇닿은 주소구역처럼 보인다. 실제적인 실현에서 다른 경우가 있을수 있는데 그것은 기억기관리방안과 조작체계가 조종구조를 조직하는 방법에 의존하게 된다.

표 3-5에서 지적된바와 같이 프로쎄스조종블로크는 구조화정보를 포함할수 있는데 여기에는 프로쎄스조종블로크의 련결을 허용하는 지시기들이 포함된다. 이렇게 앞의 장들에서 서술한 대기렬을 프로쎄스조종블로크의 련결목록으로 실현할수 있다. 실례로 그림 3-7 ㄱ의 대기구조를 그림 3-13에서 제기하고 있는것과 같이 실현할수 있다.

프로쎄스조종블로크의 역할

프로쎄스조종블로크는 조작체계에서 가장 중요한 자료구조이다. 매개 프로쎄스조종블로크는 조작체계가 요구하는 프로쎄스에 대한 모든 정보를 담고 있다. 일정작성, 자원배정, 새치기처리 및 성능감시 그리고 분석과 얽혀 진것들을 포함하는 조작체계의 매개 모듈이 가상적으로 블로크들을 읽거나 수정할수 있다. 프로쎄스조종블로크모임이 조작체계의 상태를 정의한다고 말할수 있다.

이것은 중요한 설계문제를 가르쳐 주고 있다. 조작체계안에 있는 몇개의 루틴들이 프로쎄스조종블로크의 정보에 접근할것을 요구한다. 이 표들에 대한 직접접근규정은 어렵지 않다. 매개 프로쎄스는 유일한 ID로 장비되여 있으며 이것은 조종블로크에 대한 지시기들의 표에서 첨수로 사용할수 있다. 난관은 접근이 아니라 오히려 보호이다. 두가지 문제점이 있다.

- 새치기조종기와 같은 단일루틴의 오유가 프로쎄스조종블로크를 파괴할수 있는데 이것은 영향 받은 프로쎄스들을 관리하는 체계의 능력을 파괴할수 있다.

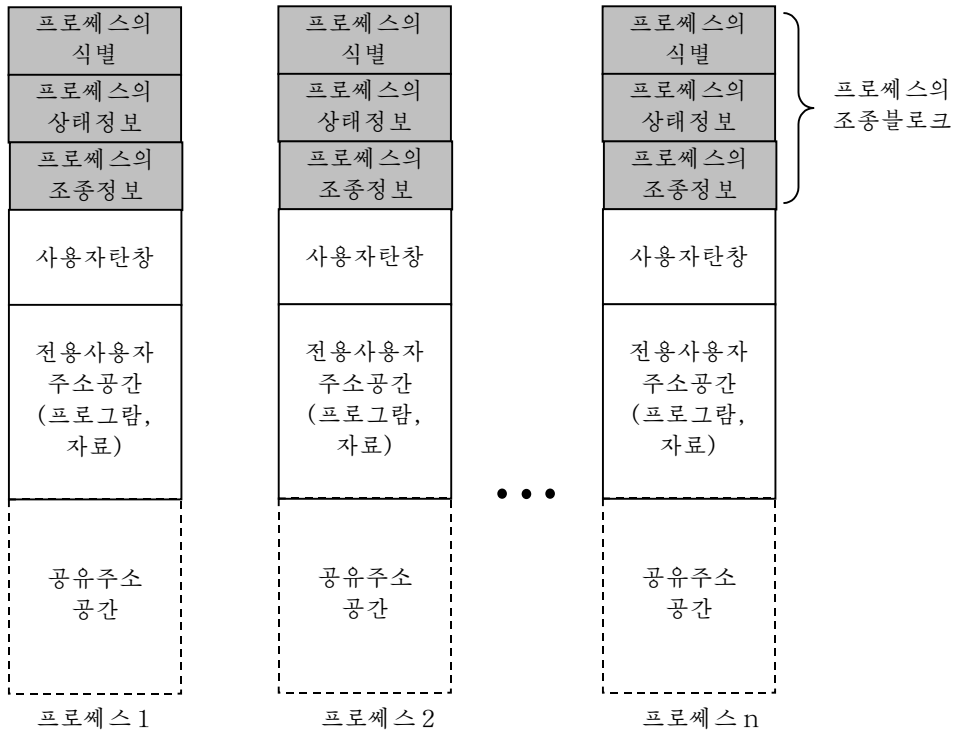



그림 3-12. 가상기억기에서 사용자프로쎄스

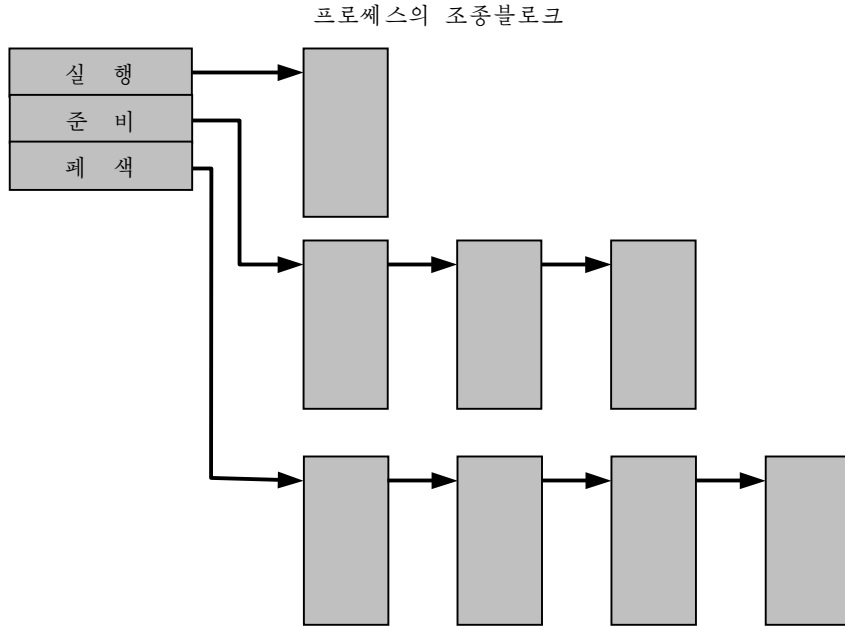



그림 3-13. 프로쎄스의 목록구조

- 프로쎄스조종블로크의 구조나 의미론에서의 설계변화는 조작체계의 많은 모듈들에 영향을 줄수 있다.

이 문제들은 조작체계에서 요구하는 모든 루틴들에 의하여 조종기루틴을 수행하도록 지정할수 있는데 그중에서 유일한 일감은 프로쎄스조종블로크를 보호하기 위한것으로서 블로크를 읽거나 쓰는데서 단독조정자의 역할을 수행한다. 그러한 루틴을 사용하는데서 성능문제와 조작체계쏘프트웨어의 나머지부분이 정확하다고 믿을수 있는 정도는 상반되는 제약관계를 가진다.

제 3 절. 프로쎄스의 조종

집행방식

조작체계가 프로쎄스를 관리하는 방법에 대한 론의를 계속하기전에 보통 조작체계와 련관된 처리기의 집행방식과 사용자프로그람과 련관된 처리기의 집행방식사이를 명백히 구별해 볼 필요가 있다. 대부분의 처리기들은 적어도 두가지 집행방식을 지원한다. 일정한 명령들은 높은 특권적방식에서만 집행될수 있다. 이것은 프로그람상태단어와 같은 조종등록기, 기본입출력명령, 기억기관리와 관련되는 명령들을 읽거나 변경시키는것을 포함한다. 게다가 주기억기의 일정한 구역은 높은 특권방식에서만 접근할수 있다.

낮은 특권방식을 흔히 **사용자**방식이라고 한다. 왜냐하면 사용자프로그람이 대체로 이 방식에서 집행되기때문이다. 높은 우선권방식에는 **체계방식**, **조종방식**, **핵심부방식**이 있다. 마지막용어는 조작체계의 핵심부를 가리키는 용어로서 이것은 중요한 체계기능들을 포함하고 있는 조작체계의 일부분이다. 표 3-7에서는 대체로 조작체계의 핵심부에서 보게 되는 기능들을 목록화하고 있다.

표 3-7. 조작체계핵심부의 대표적기능

프로세스의 관리

- 프로세스의 창조와 완료
- 프로세스의 일정작성과 할당
- 프로세스의 절환
- 프로세스의 동기화 및 프로세스사이 통신에서의 지원
- 프로세스조종블로크의 감시

기억기관리

- 프로세스에 주소공간, 기억기관리배정
- 교체
- 페지 및 토막관리

입출력관리

- 완충기관리
- 프로세스에 입출력통로 및 장치배정

지원기능

- 새치기조종
- 회계
- 감시

두가지 방식을 사용하는 리유는 명백하다. 조작체계, 프로세스조종블로크와 같은 주요조작체계표들을 사용자프로그람의 간섭으로부터 보호하는것이 중요하다. 핵심부방식에서는 쏘프트웨어가 처리기와 그의 모든 명령, 등록기 및 기억기에 대한 조종을 완료한다. 이 조종준위는 필요 없으며 안전을 위해 사용자프로그람에서는 적합하지 않다.

두가지 질문이 생긴다. 즉 어느 방식으로 집행하는가 하는것을 처리기가 어떻게 알며 그 방식을 어떻게 변화시키는가 하는것이다. 첫 질문에 대해서 말한다면 집행방식을 지적하고 있는 프로그람상태단어(PSW)에 하나의 비트가 있다. 이 비트는 일정한 사건에 응답하여 변화된다. 실례로 사용자가 조작체계를 호출할 때 방식은 핵심부방식으로 설정된다. 대체로 이것은 방식을 변화시키는 명령을 집행하여 수행된다. 이것을 수행하는 한가지 실례는 VAX상에서의 방식변경(CHM)명령이다. 사용자가 체계봉사를 호출하거나 새치기가 조종을 체계루틴에로 이송시킬 때 루틴은 CHM을 집행하여 높은 특권방식에 들어 가 그것을 집행하고 조종을 사용자프로세스에 반환하기전에 낮은 특권방식으로 다시 들어 간다. 사용자프로그람이 CHM을 집행하려고 한다면 그것은 단순히 조작체계에 대한 하나의 호출로 되는데 이것은 방식변화가 허용되지 않는 한 오유를 돌려 준다.

프로쎄스의 창조

제3장 제1절에서는 새로운 프로쎄스의 창조를 일으키는 사건들을 설명하였다. 프로쎄스와 관련된 자료구조를 설명하였으므로 이제는 실제적으로 프로쎄스를 창조하는데 포함되는 단계들을 간단히 설명하려고 한다.

일단 조작체계가 리유가 어떻든(표 3-1) 새로운 프로쎄스를 창조할것을 결심하면 그것은 다음과 같이 진행할수 있다. 즉

1. **유일한 프로쎄스식별자를 새로운 프로쎄스에 할당한다.** 이때 하나의 새로운 입구점이 초기프로쎄스표에 추가되는데 이 표는 프로쎄스당 하나의 입구점을 포함한다.

2. **프로쎄스에 공간을 배정한다.** 이것은 프로쎄스형태의 모든 요소들을 포함한다. 이렇게 하여 조작체계는 얼마만한 공간이 전용사용자주소공간(프로그람과 자료)과 사용자탄창에 요구되는가를 알아야 한다. 이 값들은 프로쎄스의 형에 따라 초기값으로 할당되거나 일감창조시 사용자요청에 기초하여 설정될수 있다. 다른 프로쎄스가 프로쎄스를 새끼친다면 부모프로쎄스는 조작체계에 요구되는 값들을 프로쎄스창조요청의 일부로서 넘겨 줄수 있다. 임의의 현존주소공간을 새 프로쎄스가 공유하면 적당한 련결이 설정되여야 한다. 끝으로 프로쎄스조종블로크를 위한 공간을 배정하여야 한다.
3. **프로쎄스조종블로크를 초기화한다.** 프로쎄스식별부분은 프로쎄스의 ID외에 부모프로쎄스의 ID와 같은 다른 해당한 ID들을 담고 있다. 처리기의 상태정보부분은 프로그람계수기(프로그람입구점에 설정한다)와 체계탄창(프로쎄스의 탄창경계들을 정하도록 설정한다.)을 제외하고는 대체로 모든 입구점들이 령으로 초기화된다. 프로쎄스조종정보는 표준초기값들외에 프로쎄스에 요청되는 속성들에 기초하여 초기화된다. 실례로 프로쎄스의 상태는 대체로 준비 또는 준비/중단으로 초기화된다. 더 높은 우선권에 대한 명확한 요청이 없는 이상 우선권을 가장 낮은 우선권의 초기값으로 설정할수 있다. 초기에 프로쎄스는 명백한 요청이 없거나 부모로부터 상속 받은것이 없는 이상 아무러한 자원(입출력장치, 파일)을 소유하지 않을수 있다.
4. **적당한 련결을 설정한다.** 실례로 조작체계련결항목으로서 매개 일정작성대기렬을 유지하고 있다면 새로운 프로쎄스를 준비 또는 준비/중단항목에 넣을수 있다.
5. **다른 자료구조를 창조하거나 확장한다.** 실례로 조작체계가 료금계산이나 성능평가를 목적으로 후에 사용될 매개 프로쎄스에 대한 체계지원계산프로그람파일을 유지할수 있다.

프로쎄스의 절환

표면상 프로쎄스의 절환기능은 간단한것처럼 보인다. 때때로 실행중인 프로쎄스가 새치기를 받으면 조작체계는 다른 프로쎄스를 실행상태에로 할당시키고 그 프로쎄스에로 조종을 절환한다. 그러나 설계상 몇가지 문제점들이 있다. 우선 어떤 사건들이 프로쎄스 절환을 촉발시키는가? 또 다른 문제점은 방식절환과 프로쎄스절환사이의 구별을 인식해야 한다는것이다. 끝으로 조작체계가 프로쎄스절환을 위한 조종에서 여러가지 자료구조에 대해 무엇을 해야 하는가?

프로쎄스절환시기

프로쎄스절환은 조작체계가 현재 실행중인 프로쎄스로부터 조종을 받은 임의의 시간에 발생할수 있다. 표 3-8은 조종을 조작체계에 넘겨 줄수 있는 가능한 사건들을 제기하고 있다.

우선 체계새치기를 고찰하자. 실제적으로 많은 체계들에서 그러하듯이 두 종류의 새치기를 구별할수 있는데 그중 하나는 간단히 새치기라고 하며 다른것은 함정이라고 한다. 전자는 외적이며 입출력조작의 충돌과 같이 현재 실행중인 프로쎄스와 무관계한 일정한 종류의 사건에 원인을 두고 있다. 후자는 위법파일접근시도와 같이 현재 실행중인 프로쎄스에서 발생되는 오유나 례외조건과 관련되여 있다. 보통 새치기를 보면 조종이 우선 새치기조종기에 이송되며 일부 기초적인 보조조작을 하고 발생한 특정한 형의 새치기와 관련되여 있는 조작체계루틴에로 이행한다. 실례를 들면 다음과 같은것들이 있다. 즉

- **시계새치기**: 조작체계는 현재 실행중인 프로쎄스가 최대허용시간만큼 집행해 왔는

가 아닌가를 판정한다. 만일 그렇다면 이 프로쎄스를 준비상태에로 절환시키고 또 다른 프로쎄스를 준비해야 한다.

- **입출력새치기**: 조작체계는 입출력동작이 발생했는가를 판정한다. 입출력동작이 한 개이상의 처리기가 기다리는 상태를 이룬다면 조작체계는 모든 대응하는 페색된 프로쎄스들을 준비상태에로 이동시켜야 한다(그리고 페색/중단프로쎄스들을 준비/중단상태에로 이동시켜야 한다.). 조작체계는 그다음 현재 실행상태에 있는 프로쎄스의 집행을 회복하겠는가 아니면 더 높은 우선권을 가지고 준비상태에 있는 프로쎄스를 처리하겠는가 하는것을 결정해야 한다.
- **기억기부재**: 처리기는 주기억기에 없는 단어에 대한 가상기억주소참조와 맞다들린다. 조작체계는 참조하는 단어를 포함하고 있는 기억블로크(페지나 토막)를 2차기억기로부터 주기억기에로 끌어 들여야 한다. 기억기의 블로크를 끌어 들이도록 입출력요청을 내보낸 다음 조작체계는 프로쎄스절환을 수행하여 또 다른 프로쎄스의 집행을 회복할수 있다. 즉 기억기부재를 가진 프로쎄스는 페색상태에 놓인다. 희망하는 블로크를 기억기에 끌어 들이면 프로쎄스는 준비상태에 놓이게 된다.

표 3-8. 프로쎄스집행의 새치기수법

기구	원인	사용
새치기	현재 명령집행에 대한 외적원인	비동기적인 외부사건에 대한 반응
함정	현재 명령의 집행과 관련	오유 또는 례외조건조종
감시기호출	명백한 요청	조작체계기능호출

함정에 대해서 말한다면 조작체계는 오유나 례외조건이 치명적인가를 판정한다. 만일 그렇다면 현재 집행중인 프로쎄스를 출구상태로 이동시키고 프로쎄스절환을 발생시킨다. 만일 그렇지 않다면 조작체계의 동작은 오유 및 조작체계의 설계특성과 관련된다. 일부 회복수속을 시도하거나 사용자에게 간단히 통지할수도 있다. 프로쎄스절환을 꼭 하거나 현재 실행중인 프로쎄스를 계속 실행할수도 있다.

끝으로 조작체계는 **감독기호출**에 의하여 집행되고 있는 프로그람으로부터 활성화될수도 있다. 실례로 사용자프로쎄스가 실행중에 있고 파일열기와 같은 입출력조작을 요청하는 명령이 집행된다. 이 호출은 조작체계코드의 일부인 루틴을 이송시킨다. 일반적으로 체계호출을 사용하면 사용자프로쎄스를 페색상태에 놓이게 한다.

방식절환

제1장에서는 명령주기의 일부로서 새치기주기가 포함된다는것을 설명하였다. 새치기주기에서 처리기는 새치기가 발생했는가를 알아 보기 위하여 검사하는데 그 새치기는 새치기신호가 있으면 통지를 받는다. 아무런 새치기도 기다리지 않는다면 처리기는 불러내기주기를 계속하여 현재프로쎄스에서 현재프로그람의 다음명령을 불러 낸다. 만일 새치기가 기다리고 있으면 처리기는 다음과 같은것을 한다.

1. 집행되고 있는 현재프로그람의 문맥을 보관한다.
2. 프로그람계수기를 새치기처리프로그람의 시작주소로 설정한다.
3. 사용자방식으로부터 핵심부방식으로 절환하여 새치기처리코드가 특권명령들을 포함할수 있게 한다.

처리기는 불러내기주기를 계속하여 새치기처리프로그람의 첫 명령을 불러 내며 이것

이 그 새치기를 봉사한다.

이제 발생할수 있는 질문은 다음과 같다. 즉 보관되는 문맥을 무엇이 구상하는가? 대답은 분명 정보를 포함시켜 새치기처리프로그람을 집행하여 변경시킬수 있으며 새치기된 프로그람을 계속하는데 필요한 어떤 정보를 포함해야 한다는것이다. 이와 같이 처리기의 상태정보라고 하는 프로쎄스조종블로크의 일부를 보관시킬수 있다. 이것은 프로그람계수기, 다른 처리기의 등록기들 및 탄창정보를 담고 있다.

그밖에 또 어떤것을 해야 하는가? 그것은 다음에 무엇이 있겠는가에 관계된다. 새치기처리기는 대체로 새치기와 관련되는 몇개의 기초적과제들을 수행하는 짧은 프로그람이다. 실례로 새치기의 존재를 알려 주는 기발이나 지시기를 재설정한다. 그것은 입출력모듈과 같은 새치기를 발행한 입구점에 응답을 보낼수 있다. 그리고 새치기를 일으킨 사건의 효과와 관련한 일정한 기초적인 보조조작을 할수도 있다. 실례로 새치기가 입출력사건과 관련이 있다면 새치기처리기는 오유조건을 검사한다. 오유가 발생했다면 새치기처리기는 원래 그 입출력조작을 요청한 프로쎄스에 신호를 보낼수도 있다. 새치기가 시계에 의한것이라면 새치기처리기는 조종을 할당기에 넘겨 주며 할당기는 조종을 다른 프로쎄스에 넘기려고 한다. 그것은 현재 집행중인 프로쎄스에 배정된 시간이 다 되였기때문이다.

프로쎄스조종블로크에서 다른 정보는 어떤가? 만일 새치기가 다른 프로쎄스에로의 절환으로 이어 진다면 일정한 작업을 수행해야 한다. 그러나 대부분의 조작체계들에서 새치기의 병행이 필수적으로 프로쎄스의 절환을 의미하지는 않는다. 가능한것은 새치기처리기가 집행한후에 현재 실행중인 프로쎄스가 집행을 계속하게 되는것이다. 그 경우에 필요한것은 새치기가 발생할 때 처리기의 상태정보를 보관하는것이며 진행중이던 프로그람에 조종이 반환될 때 그 정보를 회복하는것이다. 대체로 보관과 회복기능은 하드웨어적으로 수행된다.

프로쎄스상태의 변화

명백한것은 방식절환이 프로쎄스의 절환과 다른 개념이라는것이다.[8] 방식절환은 현재 실행상태에 있는 프로쎄스의 상태를 변화시키지 않고 발생할수도 있다. 그 경우에 문맥보관과 다음회복은 작은 보조조작을 포함한다. 그러나 현재 실행중인 프로쎄스를 다른 상태(준비, 페색 등)로 이동시킨다면 그때 조작체계는 자기의 환경에서 본질적인 변화를 일으켜야 한다. 완전한 프로쎄스절환에 포함되는 단계는 다음과 같다. 즉

1. 프로그람계수기와 다른 등록기들을 포함하여 처리기의 문맥을 보관한다.
2. 현재 실행상태에 있는 프로쎄스의 프로쎄스조종블로크를 갱신한다. 이것은 프로쎄스의 상태를 다른 상태들(준비; 페색; 준비/중단; 출구)중의 하나로 변화시킨다. 실행상태를 벗어 나는 리유와 체계지원계산프로그람정보를 포함하여 다른 관련있는 마당들도 역시 갱신해야 한다.
3. 프로쎄스의 프로쎄스조종블로크를 적당한 대기렬에로 이동시킨다(준비: 사건 *i*에 대해 페색; 준비/중단)
4. 또 다른 프로쎄스를 선택하여 집행시킨다. 이것은 제4편에서 고찰한다.
5. 선택된 프로쎄스의 프로쎄스조종블로크를 갱신한다. 이것은 프로쎄스의 상태를 실행상태로 변화시킨다.

---

[8] 문맥절환이란 말은 흔히 OS문헌이나 교과서들에서 찾아 보게 된다. 다행히 대부분의 문헌들이 여기서 프로쎄스절환이라는 의미를 보여 주기 위해 이 용어를 사용한다 하여도 다른데서는 그것을 방식절환이나 지어 스레드절환이라는 의미로 사용한다(다음장에서 정의한다.). 모호성을 피하기 위해 이 책에서는 그 용어를 사용하지 않는다.

6. 기억기관리자료구조를 갱신한다. 주소번역관리방법에 따라 이것이 요구될수도 있는데 이 문제는 제3편에서 고찰한다.
7. 프로그람계수기와 다른 등록기들의 이전값을 적재함으로써 선택된 프로쎄스가 실행상태밖으로 마지막으로 절환될 때 존재했던 처리기의 문맥을 회복한다.

이처럼 프로쎄스절환은 상태변화를 포함하여 방식절환보다 더 많은 노력을 요구한다.

조작체계의 집행

제2장에서는 조작체계들에 대하여 두개의 관통되는 사실을 지적하였다. 즉

- 조작체계기능들은 보통 콤퓨터쏘프트웨어와 같은 방식 즉 처리기가 집행하는 프로그람이다.
- 조작체계는 흔히 조종을 넘겨 주며 조작체계에 회복시켜 주는것을 처리기에 의존한다.

만일 조작체계가 프로그람들의 집합이고 그것을 다른 프로그람들과 꼭같이 처리기가 집행한다면 조작체계가 하나의 프로쎄스인가? 만일 그렇다면 그것을 어떻게 조종하는가? 이 흥미 있는 질문들은 몇가지 설계방법들을 제기하였다. 그림 3-14는 여러가지 조작체계에서 찾아 보게 되는 방법들을 분류하여 설명하고 있다.

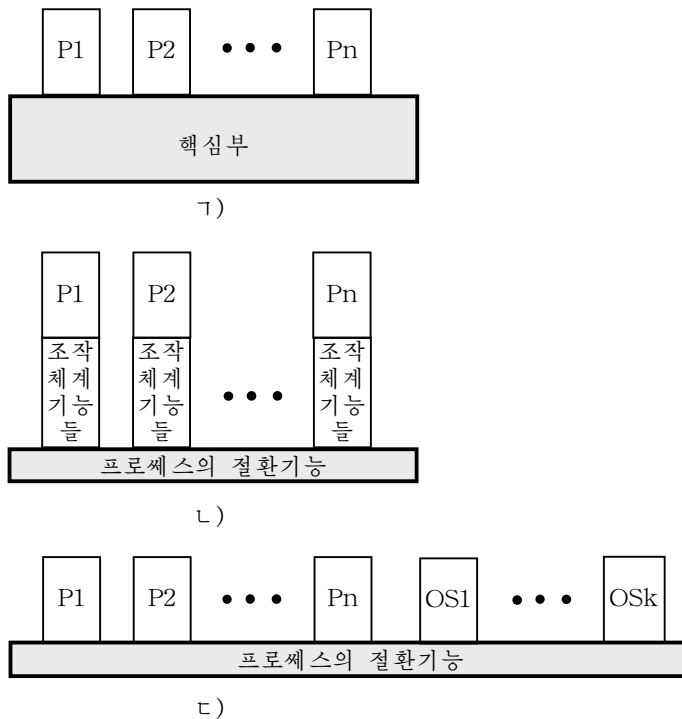



그림 3-14. 조작체계와 사용자프로쎄스사이의 관계
ㄱ-개별핵심부, ㄴ- 사용자프로쎄스에서 조작체계기능의 집행
ㄷ-개별프로쎄스로서 조작체계기능의 집행

비프로쎄스핵심부

많은 구식조작체계들에서 아주 고전적이며 공통적인 한가지 방법은 임의의 프로쎄스밖에서 조작체계의 핵심부를 집행시키는것이다(그림 3-14 ㄱ). 이 방법을 놓고 볼 때 현재 실행중인 프로쎄스가 새치기를 받거나 감시기호출을 받으면 프로쎄스의 방식문맥이 보관되며 조종은 핵심부에 넘어 가게 된다. 조작체계는 자기자체가 사용할 기억구역과 수속호출과 복귀를 위한 자기자체의 체계탄창을 가지고 있다. 조작체계는 임의의 희망하는 기능을 수행할수 있으며 새치기된 프로쎄스의 문맥을 회복하여 새치기된 프로쎄스에서 계속해 나가도록 해준다. 조작체계는 번갈아 가면서 프로쎄스의 환경을 보관하는 기능을 수행할수 있으며 다른 프로쎄스들을 일정작성 및 할당하는데 착수할수 있다. 이것이 일어 나겠는지 일어 나지 않겠는지 하는것은 그때 당시의 새치기와 문맥에서의 원인에 의존한다.

임의의 경우에 여기서 중요한것은 프로쎄스에 대한 개념을 사용자프로그람에만 적용하는것으로 고찰한다는것이다. 조작체계코드는 특권방식에서 동작하는 개별적인 입구점과 같이 집행된다.

사용자프로쎄스에서의 집행

보다 작은 기계들(PC들, 워크스테이션들)에서 조작체계에 공통인것중의 하나는 가상적으로 모든 조작체계쏘프트웨어를 사용자프로쎄스의 문맥에서 집행시키는데 있다. 그 견해는 조작체계가 기초적으로 사용자프로쎄스의 환경에서 집행되는 여러가지 기능을 수행하기 위해 사용자가 호출하는 루틴들의 집합이라는것이다. 이것을 그림 3-14 ㄴ에서 설명하고 있다. 어떤 주어 진 점에서 조작체계는 n개의 프로쎄스형태를 관리하고 있다. 매개 상은 그림 3-12에서 설명한 구역뿐아니라 핵심부프로그람을 위한 프로그람, 자료와 탄창구역을 포함한다.

그림 3-15는 이 전략에서의 대표적인 프로쎄스형태의 구조를 제기하고 있다. 개별적인 핵심부탄창은 프로쎄스가 핵심부방식에 있는동안 호출/복귀를 관리하는데 사용된다. 조작체계코드와 자료는 공유된 주소공간에 있으며 모든 사용자프로쎄스가 공유한다.

새치기, 함정 또는 감시기호출이 발생할 때 처리기는 핵심부방식에 놓이며 조종은 조작체계에 넘어 간다. 이 목적을 위해 방식문맥이 보관되며 조작체계루틴에서 방식절환이 일어 난다. 그러나 집행은 현재 사용자프로쎄스에서 계속한다. 이렇게 하여 프로쎄스절환은 수행되지 않고 같은 프로쎄스에서 방식절환만 한다.

만일 조작체계가 자기 작업을 완료하고 현재프로쎄스가 계속 실행하게 된다는것을 판정하면 그때 방식절환은 현재프로쎄스에서 새치기된 프로그람을 계속 집행한다. 이것이 이 방법의 주요우점중의 하나이다. 즉 사용자프로그람은 일정한 조작체계루틴을 사용하기 위하여 새치기됐고 그다음에 다시 집행하는데 이 모든것은 두 프로쎄스절환의 반칙을 초래하지 않고 발생하였다. 그러나 만일 이전에 집행중인 프로그람에로 복귀시키지 않으면서 프로쎄스절환이 발생한다는것을 판정하면 그때 조종은 프로쎄스절환루틴에로 넘어 간다. 이 루틴은 현재프로쎄스에서 집행할수도 있고 하지 않을수도 있는데 그것은 체계설계에 따른다. 그러나 일정한 점에서 현재프로쎄스는 비실행중상태에 놓여야 하며 다른 프로쎄스는 집행중상태로 되여야 한다. 이 단계에서는 집행이 모든 프로쎄스를 밖에서 일어 나는것으로 보는것이 론리적으로 가장 편리하다.

어떤 측면에서는 조작체계에 대한 이 견해가 아주 주목할만하다. 간단히 말하면 일정한 시점에서 프로쎄스는 자기의 상태정보를 보관하고 준비상태에 있는것들중에서 다른 프로쎄스를 선택하여 실행시키며 조종을 프로쎄스에 넘긴다. 이것이 독단적이고 실제로 무질서한 상태로 되지 않는 리유는 림계시간동안 사용자프로쎄스에서 집행되는 코드가

조작체계코드에 공유되며 사용자코드에는 공유되지 않는다는것이다. 사용자방식과 핵심부방식에 대한 개념때문에 사용자가 조작체계루틴을 함부로 다치거나 간섭할수 없다. 지어 그것들이 사용하는 프로쎄스환경에서 집행하고 있다 하여도 그렇다. 이것은 프로쎄스와 프로그람에 대한 개념사이에는 명백한 차이가 있으며 둘사이의 관계가 1 대 1이 아니라는것을 말해 준다. 프로쎄스에서 사용자프로그람과 조작체계프로그람은 둘다 집행할수 있으며 여러가지 사용자프로쎄스에서 집행하는 조작체계의 프로그람들은 동일하다.

프로쎄스에 기초한 조작체계

그림 3-14 ㄷ에서 설명한 또 다른 한가지 방법은 조작체계를 체계프로쎄스들의 집합으로 실현하는것이다. 다른데서와 마찬가지로 핵심부의 일부분인 쏘프트웨어는 핵심부방식으로 집행한다. 그러나 이 경우에 주요핵심부기능들은 개개의 프로쎄스들로 조직된다. 또한 임의의 프로쎄스밖에서 집행되는 작은 량의 프로쎄스절환코드가 있을수 있다.

이 방법은 몇가지 우점을 가지고 있다. 모듈들사이에 최소이면서 명백한 대면부를 가진 모듈화된 조작체계의 사용을 장려하는것은 일정한 프로그람설계규칙을 요구한다. 더우기 일부 비림계조작체계기능들은 개개의 프로쎄스들로 편리하게 실현된다. 실례로 이미 감시기프로그람을 언급했는데 그것은 여러가지 자원(프로쎄스, 기억기, 통로 등)의 사용률준위와 그 체계에서 사용자프로쎄스의 향상률을 기록한다. 이 프로그람이 임의의 능동프로쎄스에 특정한 봉사를 주지 않기때문에 그것은 아직 조작체계가 기동시킬수 있다. 프로쎄스와 같이 그 기능은 할당된 우선권준위에서 실행할수 있으며 할당기의 조종하에 다른 프로쎄스들과 교차처리된다. 끝으로 프로쎄스의 모임으로 조작체계를 실현하는것은 다중처리기 또는 다중콤퓨터환경에서 유용한데 그 환경에서 일부 조작체계봉사들은 전용처리기들에 떠맡겨 성능을 개선할수 있다.

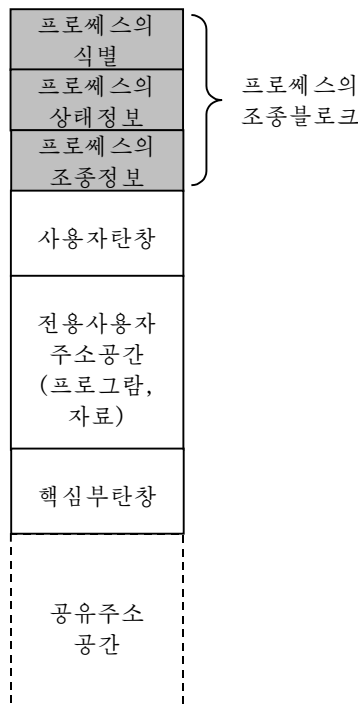



그림 3-15. 프로쎄스형태: 조작체계는 사용자공간에서 집행한다.

제 4 절. UNIX SVR4의 프로쎄스관리

UNIX체계 V는 단순하지만 강력한 프로쎄스봉사를 사용할수 있게 하며 사용자가 변경시킬수 있다. UNIX는 그림 3-14 ㄴ의 모형을 따르고 있다. 여기서는 대부분의 조작체계가 사용자프로쎄스의 환경에서 집행한다. 그러므로 두 방식 즉 사용자와 핵심부방식을 요구한다. UNIX는 두개 범주의 프로쎄스 즉 체계프로쎄스와 사용자프로쎄스를 사용한다. 체계프로쎄스는 핵심부방식에서 실행하여 기억기의 할당 및 프로쎄스교체와 같은 관리 및 보조조작기능을 수행하기 위하여 조작체계코드를 집행한다. 사용자프로쎄스는 사용자프로그람과 편의프로그람들을 집행하기 위하여 사용자방식에서 동작하며 핵심부에 속하는 명령문을 집행하기 위하여 핵심부방식에서 동작한다. 사용자프로쎄스는 어떤 례외(고장)가 발생되거나 어떤 새치기가 일어 날 때 체계호출을 하여 핵심부방식에 들어간다.

프로쎄스의 상태

UNIX조작체계는 총 9개의 프로쎄스상태를 인식하는데 이것은 표 3-9에 목록화되여 있으며 상태이행도는 그림 3-16에서 보여 주고 있다([BACH86]에 있는 그림에 기초함). 이 그림은 그림 3-7과 류사한데 두개의 폐색상태에 대응하는 두개의 UNIX잠자기상태를 가진다. 그 차이를 간단히 요약할수 있다. 즉

- UNIX는 프로쎄스가 사용자방식에서 집행중인가, 핵심부방식에서 집행중인가를 지적하기 위해 두개의 실행상태를 사용한다.
- 명백한 차이는 두개의 상태 즉 기억기에서 실행할 준비와 선취상태사이에서 이루어 진다. 이것들은 본질적으로 같은 상태이므로 점선으로 그것들을 련결하여 가리키고 있다. 그 차이는 선취된 상태가 들어 가게 되는 방법을 강조해 준다. 프로쎄스가 핵심부방식에서 실행하고 있을 때(감독기호출, 시계새치기 또는 입출력새치기의 결과로) 핵심부가 자기의 작업을 완료하고 조종을 사용자프로그람에로 복귀시킬 준비가 되는 때가 온다. 이 시점에서 핵심부는 준비되여 있고 보다 높은 우선권을 가지고 있는것에 유리하게 현재의 프로쎄스를 선취하도록 할수 있다. 그 경우에 현재프로쎄스는 선취된 상태에로 이동한다. 그러나 할당을 목적으로 선취된 상태에 있는 프로쎄스들과 기억기에서 실행할 준비가 된 프로쎄스들은 하나의 대기렬을 형성한다.

선취는 프로쎄스가 핵심부방식으로부터 사용자방식으로 이동할 때에만 발생할수 있다. 프로쎄스가 핵심부방식에서 실행하고 있는 동안은 그것을 선취할수 없다. 이것은 UNIX로 하여금 실시간처리에 대처할수 없게 한다. 실시간처리의 요구에 대한 론의는 제10장에서 한다.

두개의 프로쎄스가 UNIX에서 독특하다. 프로쎄스 0은 체계가 첫 넣기할 때 발생하는 특수한 프로쎄스이다. 실제적으로 그것은 첫 넣기시기에 적재되는 자료구조로 미리 정의된다. 그것이 바로 교체프로그람프로쎄스이다. 더우기 프로쎄스 0은 초기화프로쎄스라고 하는 프로쎄스 1을 새끼치는데 체계에서 모든 다른 프로쎄스들은 선조로서 프로쎄스 1을 가진다. 새로운 대화형가입자가 체계에 가입할 때 사용자용으로 사용자프로쎄스를 창조하는것이 바로 프로쎄스 1이다. 다음에 사용자프로쎄스로 갈래나무의 자식프로쎄스를 창조할수 있으며 그 결과 임의의 특정한 응용프로그람을 몇개의 관련 있는 프로그람으로 구성할수 있다.

프로쎄스의 서술

UNIX에서 프로쎄스는 조작체계에 프로쎄스들을 관리 및 할당하는데 필요한 모든 정보를 주는 보다 복잡한 자료렬의 모임으로 된다. 표 3-10은 프로쎄스형태의 요소들을 요약하고 있는데 이것은 세개의 부분 즉 사용자준위문맥, 등록기문맥 및 체계준위문맥으로 조직된다.

사용자준위문맥은 사용자프로그람에 대한 기초적요소들을 담고 있으며 번역된 목적파일로부터 직접 발생될수 있다. 사용자프로그람은 본문과 자료구역으로 분리된다. 즉 자료구역은 읽기전용으로 되며 프로그람의 명령으로 간주된다. 프로쎄스가 집행하고 있는 동안 처리기는 수속호출과 복귀 및 파라메터넘기기를 위해 사용자탄창구역을 사용한다. 공유기억구역은 다른 프로쎄스와 공유되는 자료구역이다. 하나의 공유기억구역에 대한 물리적복사뿐아니라 가상기억기에 대한 사용자의 복사도 있을수 있는데 그것은 마치 공유기억구역이 주소공간에 있는 매개 프로쎄스를 각자가 공유하고 있는듯하다. 프로쎄스가 실행중이 아닐 때 처리기의 상태정보는 **등록기문맥**구역에 기억된다.

체계준위문맥은 조작체계가 프로쎄스를 관리하기 위하여 요구하는 나머지 정보를 담고 있다. 그것은 정적부분(크기에서 고정되여 있고 수명전기간 프로쎄스와 함께 있다.)과 동적부분(프로쎄스의 수명기간 크기가 변한다.)으로 구성되여 있다. 정적부분중에서 하나의 요소는 프로쎄스의 표입구점이다. 이것은 실제상 조작체계가 유지하는 프로쎄스표의 일부로서 프로쎄스마다 하나의 입구점을 가진다. 프로쎄스표입구점은 항상 핵심부에 접근할수 있는 프로쎄스조종정보를 담고 있다. 이로부터 가상기억기체계에서 모든 프로쎄스표입구점들은 주기억기에 유지되여 있다. 표 3-11은 프로쎄스표입구점의 내용을 목록화하고 있다. 사용자구역 즉 U구역은 추가적인 프로쎄스조종정보를 담고 있는데 이것은 프로쎄스의 문맥에서 집행하고 있을 때 핵심부가 요구하게 된다. 그것은 또한 프로쎄스들을 기억기에로 또는 기억기로부터 페지화할 때 사용할수 있다. 표 3-12는 이 표의 내용을 보여 주고 있다.

표 3-9. UNIX 프로쎄스의 상태

사용자의 실행중	사용자방식에서 집행중
핵심부의 실행중	핵심부방식에서 집행중상태
기억기에서 실행준비	기억기핵심부가 그것을 일정작성하자마자 실행할 준비상태
기억기에서 잠자는 상태	사건이 발생할 때까지 집행할수 없는 상태. 이때 프로쎄스는 주기억기에 있다(페색된 상태).
실행준비 및 교체된 상태	프로쎄스가 실행할 준비는 되였지만 핵심부가 그것을 집행하도록 일정작성하기전에 교체프로그람이 그 프로쎄스를 주기억기에로 교체해 넣어야 한다.
잠자면서 교체된 상태	프로쎄스가 사건을 기다리고 있으며 2차기억기와 교체되였다(페색된 상태).
선취된 상태	프로쎄스가 핵심부방식으로부터 사용자방식으로 복귀하고 있지만 핵심부는 그것을 선취하고 프로쎄스절환을 하여 다른 프로쎄스를 일정작성한다.
창조된 상태	프로쎄스가 새롭게 창조되고 아직 실행할 준비는 되여 있지 않다.
좀비	프로쎄스가 더이상 존재하지 않지만 그의 부모프로쎄스가 수집하도록 레코드를 남겨 둔다.

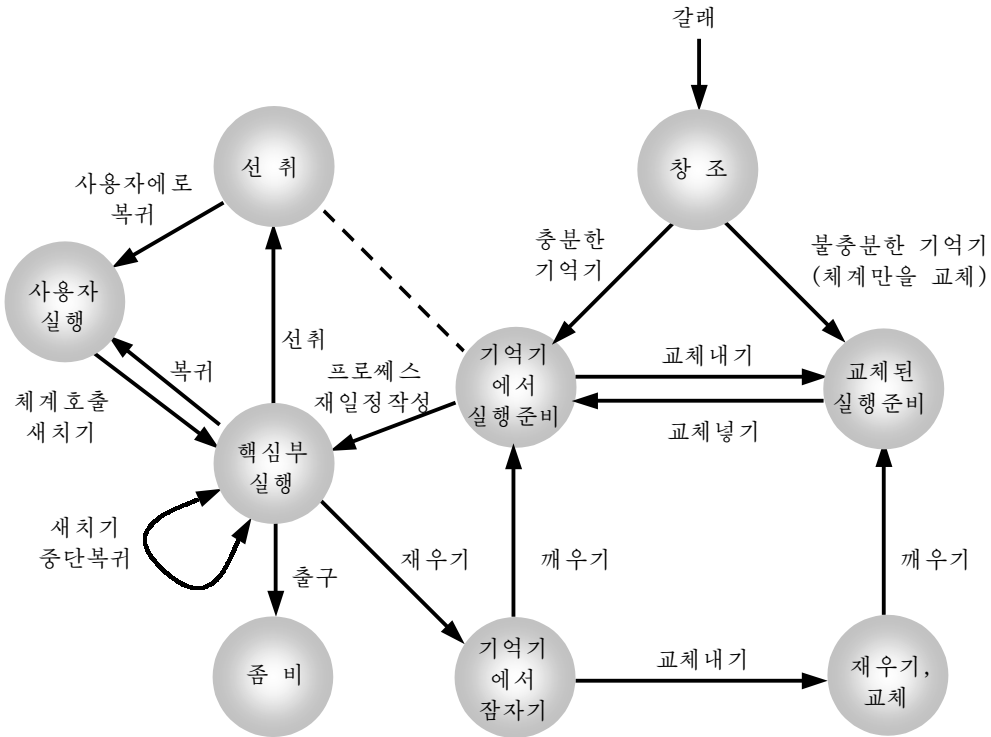



그림 3-16. UNIX 프로쎄스의 상태이행도

프로쎄스표입구점과 U구역의 명백한 차이는 UNIX핵심부가 항상 일정한 프로쎄스의 문맥내에서 집행한다는 사실을 반영한다. 많은 시간동안 핵심부는 프로쎄스와 관련된 것을 취급한다. 그러나 핵심부가 다른 프로쎄스들을 할당하는데서 예비적인 일정작성알고리듬을 수행하고 있을 때와 같이 그 시간의 일부만 다른 프로쎄스에 대한 정보에 접근할것을 요구하게 된다.

체계준위문맥의 셋째 정적부분은 기억기관리체계가 사용하는 프로쎄스별구역표이다. 끝으로 핵심부탄창은 체계준위문맥의 동적부분이다. 프로쎄스가 핵심부방식에서 집행하고 있을 때 이 탄창을 사용하며 이 탄창은 수속호출과 새치기가 발생할 때 보관하고 회복해야 할 정보를 담고 있다.

프로쎄스의 조종

UNIX에서 프로쎄스의 창조는 핵심부체계호출 fork( )에 의해 이루어 진다. 프로쎄스가 갈래 (fork) 요청을 내보낼 때 조작체계는 다음과 같은 기능을 수행한다[BACH 86]. 즉

1. 새로운 프로쎄스에서 프로쎄스표안에 홈을 배정한다.
2. 자식프로쎄스에 유일한 프로쎄스의 ID를 할당한다.
3. 임의의 공유기억기를 제외하고 부모프로쎄스형태를 복사한다.
4. 부모가 소유하고 있는 파일계수기를 증가시켜 추가적인 프로쎄스가 이제 그 파일을 또 소유한다는것을 반영한다.
5. 자식프로쎄스를 실행할 준비가 된 상태에로 할당한다.
6. 부모프로쎄스에 자식의 ID수자를 돌려 주며 자식프로쎄스에는 0값을 준다.

표 3-10. UNIX 프로쎄스형태

사용자준위문맥	
프로쎄스의 본문	프로그람중에서 집행할수 있는 기계명령들
프로쎄스의 자료	이 프로쎄스의 프로그람이 접근할수 있는 자료
사용자탄창	사용자방식에서 집행하는 기능들에서의 독립변수, 국부변수와 지시기들을 담고 있다.
공유기억기	다른 프로쎄스와 공유된 기억기이며 프로쎄스사이 통신에 사용한다.
등록기문맥	
프로그람계수기	집행할 다음명령의 주소; 이 프로쎄스의 핵심부가 사용자기억공간에 있을수 있다.
처리기의 상태등록기	선취될 때 하드웨어상태를 담고 있다. 즉 내용과 서식이 하드웨어에 의존한다.
탄창	핵심부나 사용자탄창의 앞점을 가리키는데 그 시기나 선취에서의 조작방식에 의존한다.
일반목적등록기들	하드웨어에 의존
체계준위문맥	
프로쎄스의 표입구점	프로쎄스의 상태를 정의한다. 이 정보는 조작체계에 항상 접근할수 있다.
U(사용자)구역	프로쎄스의 문맥에서만 접근할것을 요구하는 프로쎄스의 조종정보
프로쎄스별구역표	가상주소로부터 물리주소에로의 사영을 정의한다. 또한 프로쎄스에 허용되는 접근의 형 즉 읽기전용, 읽기쓰기 또는 읽고 집행을 가리키는 허용마당을 담고 있다.
핵심부탄창	프로쎄스가 핵심부방식에서 집행할 때 핵심부수속들의 탄창를을 담고 있다.

이 모든 작업은 부모프로쎄스에서 핵심부방식으로 수행된다. 핵심부가 이 기능을 완료할 때 그것은 할당기루틴의 일부로서 다음의것들중의 하나를 할수 있다.

1. 부모프로쎄스에 머무른다. 조종은 부모의 갈래호출점에서 사용자방식으로 복귀한다.
2. 조종을 자식프로쎄스에 이송한다. 자식프로쎄스는 부모와 같은 코드의 같은 점에서 즉 갈래호출로부터 복귀점에서 집행하기 시작한다.
3. 조종을 다른 프로쎄스에 이송한다. 부모와 자식이 둘다 실행할 준비상태에 남아있게 된다.

이 프로쎄스창조방법을 보일수 있게 하기는 좀 힘들다. 그것은 부모와 자식이 둘다 같은 구절의 코드를 집행하고 있기때문이다. 그 차이는 이것이다. 즉 갈래로부터의 복귀가 발생할 때 복귀파라메터를 검사한다. 만일 그 값이 령이면 이것은 자식프로쎄스이고 적당한 사용자프로그람이 집행을 계속해 나가도록 갈래프로그람을 집행할수 있다. 만일 그 값이 령이 아니면 이것은 부모프로쎄스이며 기본방향의 집행을 계속할수 있다.

표 3-11. UNIX프로쎄스의 표입구점

프로쎄스의 상태	프로쎄스의 현 상태
	사용자구역과 프로쎄스의 기억기구역(본문, 자료, 탄창)에 대한 지시기
프로쎄스의 크기	조작체계가 얼마만한 공간을 프로쎄스에 배정하겠는가를 알수 있게 한다.
사용자식별자	**실제사용자** ID는 실행중인 프로쎄스에 책임이 있는 사용자를 식별한다. **효과적인 사용자** ID를 사용하여 프로쎄스가 특정한 프로그람과 관련된 림시적특권을 얻을수 있다. 프로그람이 프로쎄스의 일부로 집행되고 있는 동안 프로쎄스는 효과적인 사용자 ID를 가지고 조작한다.
프로쎄스식별자	프로쎄스의 ID, 부모프로쎄스의 ID. 이것들은 프로쎄스가 갈래체계호출기간에 창조된 상태에 들어 갈 때 설정된다.
사건서술자우선권	프로쎄스가 잠자는 상태에 있을 때 유효하다. 사건이 발생할 때 프로쎄스는 실행할 준비상태로 이송된다.
우선권	프로쎄스의 일정작성에 쓰인다.
신호	프로쎄스에 보냈지만 아직 조종되지 않은 신호들을 렬거한다.
시계	프로쎄스의 집행시간, 핵심부의 자원사용 및 사용자가 설정한 시계들을 포함하는데 프로쎄스에 경보신호를 보내는데 사용한다.
P련결	준비상태의 대기렬에서 다음련결에 대한 지시기(프로쎄스가 진행할 준비가 되면 유효하다.)
기억기의 상태	프로쎄스형태가 주기억기에 있는가 또는 교체되여 나갔는가를 가리킨다. 만일 주기억기에 있으면 이 마당은 또한 그것이 교체되여 나갈수 있는가 아니면 주기억기에 림시 고정되여 있는가를 가리킨다.

표 3-12. UNIX U령역

프로쎄스표	U령역에 해당한 입구점을 가리킨다.
사용자식별자	실제적이며 효과적인 사용자 ID. 사용자특권을 결정하는데 쓰인다.
시계	프로쎄스(그리고 그의 자손들)가 사용자방식과 핵심부방식에서 집행하는데 소비한 시간을 기록한다.
신호조종기배렬	체계에서 정의된 매개 신호류형에 대하여 프로쎄스가 신호(지정된 사용자의 기능을 출구, 무시, 집행)의 접수에 어떻게 반응하는가를 가리킨다.
조종말단	존재한다면 프로쎄스에서의 가입말단을 가리킨다.
오유마당	체계호출기간에 만나는 오유를 기록한다.
귀환값	체계호출결과를 담고 있다.
입출력파라메터	이송시키려는 자료의 량. 사용자공간에서의 원천(또는 목표)자료배렬의 주소, 입출력에 대한 파일편위주소를 서술한다.
파일파라메터	현재등록부와 현재뿌리등록부가 프로쎄스의 파일체계환경을 서술한다.

표계속

사용자파일서술자표	프로쎄스가 열어 놓은 파일을 기록한다.
제한마당	그것이 쓸수 있는 프로쎄스의 크기와 파일크기를 제한한다.
허락방식마당	프로쎄스가 창조하는 파일에 대한 방식설정을 마스크한다.

요약, 기본용어 및 복습문제

현대조작체계에서 가장 기본적인 구성요소는 프로쎄스이다. 조작체계의 원리적기능은 프로쎄스의 창조, 관리 및 완료이다. 프로쎄스들이 능동상태일 때 조작체계는 매개 프로쎄스들이 처리기가 집행하는데서 배정 받은 시간을 알아야 하며 그것들의 동작을 일치시키고 상반되는 요구를 관리하며 체계자원을 프로쎄스에 배정하여야 한다.

프로쎄스관리기능을 수행하기 위하여 조작체계는 프로쎄스가 집행하는 주소공간과 프로쎄스조종블로크를 포함하는 매개 프로쎄스나 프로쎄스형태의 서술을 유지한다. 후자는 프로쎄스의 현존상태, 프로쎄스에 배정된 자원을 포함하여 프로쎄스를 관리하기 위해 조작체계가 요구하는 모든 정보를 포함하지만 전자는 다른 적합한 자료를 포함한다.

프로쎄스는 자기의 수명기간에 여러 상태사이에서 이동한다. 이 상태들중에서 가장 중요한것은 준비, 실행 그리고 폐색이다. 준비프로쎄스는 현재 비실행중이지만 조작체계가 동작하자마자 실행될 준비가 된 프로쎄스이다. 실행프로쎄스는 현재 처리기가 실행하고 있는 프로쎄스이다. 다중처리기체계에서 한개이상의 프로쎄스가 이 상태에 놓인다. 폐색프로쎄스는 입출력동작과 같은 어떤 사건의 완료를 기다리는 프로쎄스이다.

실행프로쎄스는 프로쎄스밖에서 일어나고 처리기가 인식하는 사건인 새치기에 의하여 또는 조작체계에 대한 감독기호출을 집행할 때 새치기된다. 두 경우에 처리기는 방식을 절환하여 조작체계에 조종을 이송한다. 필요한 작업을 끝낸후 조작체계는 새치기프로쎄스나 일부 다른 프로쎄스에로 절환될수 있다.

기본용어

폐색상태	특권방식	실행상태	과제
자식프로쎄스	프로쎄스	중단상태	추적
출구상태	프로쎄스조종블로크	교체조작	함정
새치기	프로쎄스형태	체계방식	사용자방식
핵심부방식	프로그람상태단어	부모프로쎄스	프로쎄스절환
방식절환	순환법	선취	준비상태
새상태			

복습문제

1. 명령수속이란 무엇인가?
2. 프로쎄스의 창조에 쓰이는 일반사건들은 무엇인가?
3. 그림 3-5의 처리모형에서 매개 상태를 간단히 정의하시오.
4. 프로쎄스를 선취한다는것은 어떤 의미인가?
5. 교체조작이란 무엇이며 그의 목적은 무엇인가?
6. 그림 3-8 ㄴ는 왜 두개의 폐색상태를 가지는가?
7. 중단프로쎄스의 네가지 특성을 렬거하시오.

8. 조작체계가 어떤 류형의 개체에 대하여 관리목적을 위한 정보표를 가지는가?
9. 프로쎄스조종블로크에서 두가지 일반적인 정보목록을 렬거하시오.
10. 왜 두가지 방식(사용자와 핵심부)이 요구되는가?
11. 새로운 프로쎄스를 창조하기 위해 연산체계가 수행하는 단계는 무엇인가?
12. 새치기와 함정사이의 차이점은 무엇인가?
13. 새치기에 대한 3가지 실례를 드시오.
14. 방식절환과 프로쎄스절환사이의 차이점은 무엇인가?

참 고 문 헌

제2장 제9절에서 서술된 책들은 모두 이 장의 자료들을 포함하고 있다. UNIX프로쎄스관리에 대한 중요한 서술은 [GOOD94]와 [GRAY97]에 있다. [NEHM75]는 프로쎄스의 할당에 요구되는 프로쎄스상태 및 조작체계의 기본지령에 대해 흥미 있게 서술하고 있다.

GOOD94 Goodheart, B., and Cox, J. *The Magic Garden Explained: The internals of UNIX System V Release 4.* Englewood Cliffs, NJ: Prentice Hall, 1994.

GRAY97 Gray, J. *Interprocess Communications in UNIX: The Nooks and Crannies.* Upper Saddle River, NJ: Prentice Hall, 1997.

NEHM75 Nehmer, J. "Dispatcher Primitives for the Construction of Operating System Kernels." *Acta Informatica,* vol. 5, 1975.

련 습 문 제

1. 프로쎄스관리에 대한 조작체계의 다섯가지 주요동작을 이름 짓고 왜 매개 동작이 요구되는가를 간단히 서술하시오.
2. [PINK89]에서 다음의 상태들이 프로쎄스용으로 정의되였다. 즉 집행(실행), 활성화(준비), 폐색 및 중단이다. 만일 프로쎄스가 자원사용을 허락할 때까지 기다린다면 프로쎄스는 폐색프로쎄스이고 그것이 이미 획득한 자원에서 조작이 끝나기를 기다린다면 중단프로쎄스이다. 많은 조작체계들에서 이 두가지 상태는 이 장에서 사용한 정의와 같이 폐색상태와 중단상태로 짝을 이룬다. 두 정의의 상대적우점들을 비교하시오.
3. 그림 3-8 ㄴ의 일곱가지 상태의 프로쎄스모형에 대하여 그림 3-7 ㄴ와 같이 대기선도를 그리시오.
4. 그림 3-8 ㄴ의 상태이행도를 고찰하자. 조작체계가 프로쎄스를 할당할 시간이며 준비상태와 준비/중단상태사이에 프로쎄스가 있으며 준비/중단상태에 있는 적어도 하나가 준비상태에서의 어떤 프로쎄스보다 더 높은 일정작성우선권을 가진다고 가정하자. 두가지 극단한 방법이 있다.

   ㄱ) 항상 교체조작을 최소화하기 위해 준비상태에 있는 프로쎄스로부터 할당하며
   ㄴ) 교체조작이 필요 없을 때 교체한다 할지라도 항상 제일 높은 우선권을 가진 프로쎄스에 우선권을 부여한다. 우선권과 성능사이의 관계를 평형시키기 위

한 중간적인 방법을 제기하시오.

5. 표 3-13은 VAX/VMS조작체계의 프로쎄스상태를 보여 준다.
   ㄱ) 그렇게 많은 명백한 기다림상태의 존재를 정식화할수 있는가?
   ㄴ) 다음 상태들은 왜 상주 및 교체되여 나간 판본들을 가지지 않는가? 즉 페지부재기다림, 충돌페지기다림, 공통사건기다림, 자유페지기다림 및 자원기다림과 같은 상태들인 경우가 그렇다.
   ㄷ) 상태이행도를 그리고 동작 즉 매개 이행이 일어 나는 원인을 설명하시오.

표 3-13. VAX/VMS프로쎄스상태

현재 집행중	실행중 프로쎄스
계산가능(상주)	주기억기에서의 준비 및 상주
계산가능(교체되여 나감)	준비. 그러나 주기억기밖으로 교체되여 나감
페지부재기다림	프로쎄스가 주기억기에 없는 페지를 참고하였고 페지읽기를 기다려야 한다.
충돌페지기다림	프로쎄스가 다른 프로쎄스에서 현재 페지부재기다림을 일으킨 공유페지 또는 프로쎄스가 읽거나 쓰는 전용페지를 참조하였다.
공통사건기다림	공유식사건기발기다림중(사건기발은 단일비트프로쎄스사이의 신호수법)
자유페지기다림	주기억기의 자유페지가 프로쎄스에 부여된 주기억기의 페지집합에 추가되는것을 기다리고 있는 층
동면기다림(상주)	프로쎄스가 자기자체를 기다림상태에 넣는다.
동면기다림(교체되여 나감)	기다리는 프로쎄스가 주기억기밖으로 교체되여 나간다.
국부사건기다림(상주)	주기억기의 프로쎄스이며 국부사건기발기다림중(보통 입출력완료)
국부사건기다림(교체되여 나감)	국부사건을 기다리고 있는 프로쎄스가 주기억기밖으로 교체되여 나간다.
중단된 기다림(상주)	프로쎄스가 다른 프로쎄스에 의해 기다림상태에 놓인다.
중단된 기다림(교체되여 나감)	중단된 프로쎄스는 주기억기밖으로 교체된다.
자원기다림	여러가지 체계자원을 기다리는 프로쎄스

6. VAX/VMS조작체계는 네개의 처리기접근방식을 사용하여 프로쎄스사이에서 체계자원의 보호와 공유를 편리하게 해준다. 접근방식은 다음과 같은것들을 결정한다. 즉
   - 명령집행권 : 처리기가 집행할수 있는 명령
   - 기억기접근권 : 현재 명령이 접근할수 있는 가상기억기의 위치

   네가지 방식은 다음과 같다.
   - 핵심부 : VMS조작체계의 핵심부를 집행하는데 그 조작체계는 기억기관리, 새치기조종 및 입출력조작을 한다.
   - 집행부 : 파일 및 레코드(디스크 및 테프)관리루틴을 포함하는 다른 조작체계호출을 집행한다.
   - 감시기 : 사용자명령에 대한 응답과 같은 다른 조작체계봉사를 진행한다.

• 사용자 : 사용자프로그람 및 콤파일러, 편집기, 련결프로그람 및 오유수정기와 같은 편의프로그람들을 집행한다.

낮은 특권방식에서 집행하는 프로쎄스는 높은 특권방식에서 집행하는 수속호출을 요구한다. 실례로 사용자프로그람이 조작체계봉사를 요구한다. 이 호출은 새로운 접근방식에서 조종을 루틴에 넘기는 새치기를 일으키는 방식변화(CHM)명령을 사용하여 수행된다. 복귀는 REI(례외 또는 새치기으로부터의 복귀)명령집행에 의하여 이루어 진다.

ㄱ) 많은 조작체계는 핵심부와 사용자의 두가지 방식을 가진다. 두가지 대신에 네가지 방식을 가지는것이 가지는 우점과 결함은 무엇인가?
ㄴ) 네개이상의 방식을 주는 경우도 있을수 있는가?

7. 앞의 문제에서 론의된 VMS방안은 그림 3-17에서 서술한바와 같이 흔히 고리보호구조라고 한다. 실지 제3장 제3절에서 서술한바와 같이 간단한 핵심부/사용자방안은 두개 고리의 구조이다. [SILB98]은 이 방법에 대한 문제를 보여 주고 있다. 고리(계층)구조의 기본결함은 그것이 알 필요의 원리를 시행하지 못하게 한다는것이다. 특히 객체가 D*j*령역에 접근할수 없다면 $j < i$여야 한다. 그러나 이것은 D*i*에서 접근할수 있는 매개 토막이 D*j*에서도 접근할수 있다는것을 의미한다.

ㄱ) 앞에서 언급한 문제를 명백히 설명하시오.
ㄴ) 고리구조식조작체계가 이 문제를 취급할수 있는 방법을 제기하시오.

8. 그림 3-7 ㄴ는 프로쎄스가 한번에 오직 한개 사건대기렬에 있다는것을 보여 준다.

ㄱ) 프로쎄스가 같은 시간에 한개이상의 사건을 기다리도록 할수 있는가? 실례를 드시오.
ㄴ) 그 경우에 새로운 구조를 구축하기 위해 대기구조그림을 어떻게 변경하겠는가 ?

9. 초기 몇가지 콤퓨터에서 새치기는 등록기값들을 주어 진 새치기신호와 련관된 고정위치에 기억시키도록 한다. 어떤 경우에 이것이 실천적으로 가능한가 ? 일반적으로 왜 그것이 불합리한가를 설명하시오.

10. 핵심부방식에서 집행하는 프로쎄스를 선취할수 없기때문에 UNIX가 실시간응용에 적합하지 않다는것을 제3장 제4절에서 설명하였다. 교정하시오.

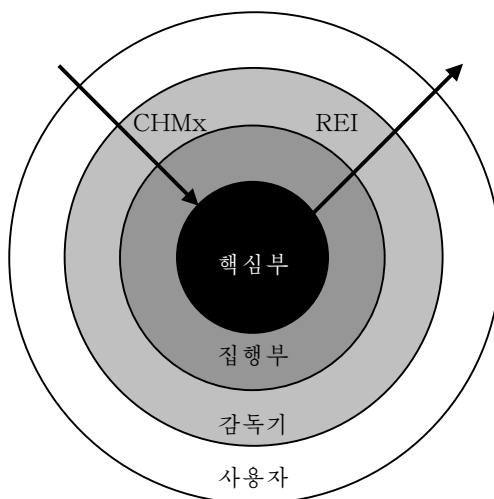



그림 3-17. VAX/VMS 접근방식

제 4 장. 스레드, SMP 및 마이크로핵심부

이 장에서는 현대조작체계에서 나서는 프로쎄스관리와 관련된 보다 고급한 개념들을 고찰한다. 먼저 프로쎄스의 개념은 지금까지 고찰한것보다 더 복잡하고 미묘하며 사실상 두가지의 독립적인 개념 즉 자원소유권과 관련된것과 집행과 관련된것으로 갈라 진다. 이 구별은 일부 조작체계에서 **스레드**구조로 발전해 왔다. 스레드를 론의한후 **대칭형다중처리**(SMP)를 본다. SMP로 조작체계는 다중처리기상에서 동시에 서로 다른 프로쎄스를 일정작성하게 된다. 마지막으로 프로쎄스관리와 다른 과제를 위한 조작체계를 구성하는데서 효과적의미를 가지는 마이크로핵심부의 개념을 소개한다.

제 1 절. 프로쎄스와 스레드

지금까지는 두가지 기능을 구체화하여 프로쎄스의 개념을 주었다. 즉

- **자원소유권**: 프로쎄스는 프로쎄스형태를 보존하는 가상주소공간을 가지며 때로는 조종이나 주기억기, 입출력통로, 입출력장치 및 파일과 같은 자원소유권을 배정받을수 있다. 조작체계는 보호기능을 수행하여 자원에 관한 프로쎄스들사이의 간섭을 방지한다.
- **일정작성/집행**: 프로쎄스의 집행은 하나이상의 프로그람을 통해 집행경로(추적)의 뒤를 따른다. 이 집행은 다른 프로쎄스의 집행과 교차처리될수도 있다. 따라서 프로쎄스는 집행상태(실행, 준비 기타)와 할당우선도를 가지며 조작체계가 일정작성하고 할당하는 실체이다.

대부분의 조작체계들에서 이 두가지 기능은 실제상 프로쎄스의 본질을 이룬다. 그러나 일부 견해는 이 두가지 기능이 독립이며 조작체계가 독립적으로 취급할수 있다는 것을 독자들에게 납득시켜야 한다. 이것은 여러 조작체계들 특히 최근에 개발된 체계에서 그렇다. 이 두가지 기능을 구분하기 위하여 할당중인 부분을 보통 **스레드** 또는 **경량프로쎄스**라고 하며 자원소유권부분을 보통 **프로쎄스**나 **과제**라고 한다.[1]

다중스레드처리

다중스레드처리는 단일프로쎄스에서 다중스레드의 집행을 보장하는 조작체계의 능력을 가리키는 말이다. 스레드의 개념을 인식하지 못한 프로쎄스당 단일스레드를 집행하는 전통적인 방법을 단일스레드방법이라고 한다. 그림 4-1의 왼쪽 절반에 보여 준 두개의 배렬이 단일스레드법이다. 그림4-1의 오른쪽 절반은 다중스레드법을 보여 준다. MS-DOS 는 단일사용자프로쎄스 및 단일스레드를 지원하는 조작체계의 한 실례로 된다. Java의 실행시 환경이 다중스레드를 가진 한개 프로쎄스체계의 실례로 된다. 이 절에서 흥미 있는것은 그 매개가 다중스레드를 지원하는 다중프로쎄스를 사용하는것이다. Windows 2000(W2K), Solaris, Linux, Mach, OS/2, 다른것들속에서 이 방법을 받아 들

---

1. 이 정도의 일관성도 보존할수 없다는것은 유감스러운 일이다. IBM의 일반형콤퓨터조작체계인 OS/390에서 주소공간과 과제에 대한 개념은 각각 이 절에서 설명하는 프로쎄스라는 용어를 (1) 용어 스레드와 등가인것으로, (2) 핵심부준위스레드로 알려 진 특정한 형의 스레드로, (3) Solaris의 경우에는 사용자준위스레드를 핵심부준위스레드로 배치하는 입구점으로 사용하고 있다.

이고 있다. 이 절에서는 다중스레드처리의 일반서술을 준다. 즉 W2K, Solaris, Linux 방법에 대한 세부들은 이 장에서 후에 론의한다.

다중스레드환경에서는 프로쎄스를 자원배정의 단위 및 보호단위로 정의한다. 다음과 같은것들이 프로쎄스와 련관되여 있다. 즉

- 프로쎄스형태를 보존하는 가상주소공간
- 처리기, 다른 프로쎄스들(프로쎄스사이 통신), 파일 그리고 입출력자원(장치와 통로)에 대한 보호접근

프로쎄스에 하나이상의 스레드가 있을수 있으며 매개는 다음과 같은것들을 가지고 있다.

- 스레드집행상태(실행, 준비 등)
- 비실행시 보관된 스레드문맥; 스레드를 프로쎄스에서 독립적인 프로그람계수기조작과 같이 보는 한가지 방법이다.
- 집행탄창
- 국부변수용스레드당 몇개의 정적기억
- 프로쎄스안에서 모든 다른 스레드와 공유된 프로쎄스의 기억기와 자원에 대한 접근

그림 4-2는 프로쎄스관리측면에서 스레드와 프로쎄스사이의 차이를 보여 준다. 단일스레드처리프로쎄스모형(즉 스레드에 대한 명백한 개념이 전혀 없다.)에서 프로쎄스는 프로쎄스집행의 호출/복귀동작을 관리하는 사용자 및 핵심부탄창은 물론 프로쎄스조종블로크와 사용자주소공간을 포함한다. 프로쎄스가 실행중인 동안 처리기의 등록기는 프로쎄스에 의해 조종되며 등록기의 내용은 프로쎄스가 비실행중일 때 보관된다. 다중스레드

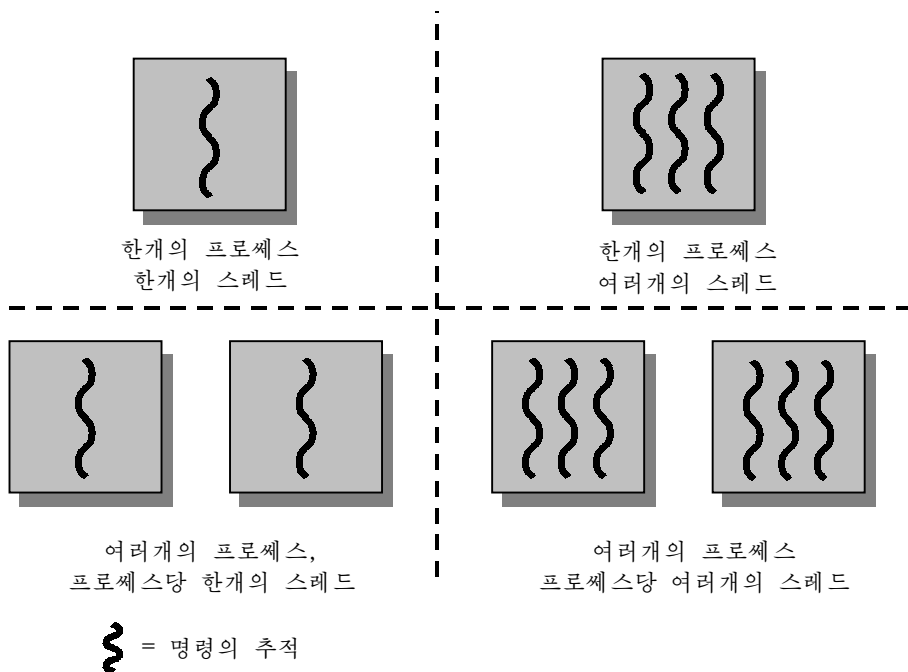



그림 4-1. 스레드와 프로쎄스[ANDE97]

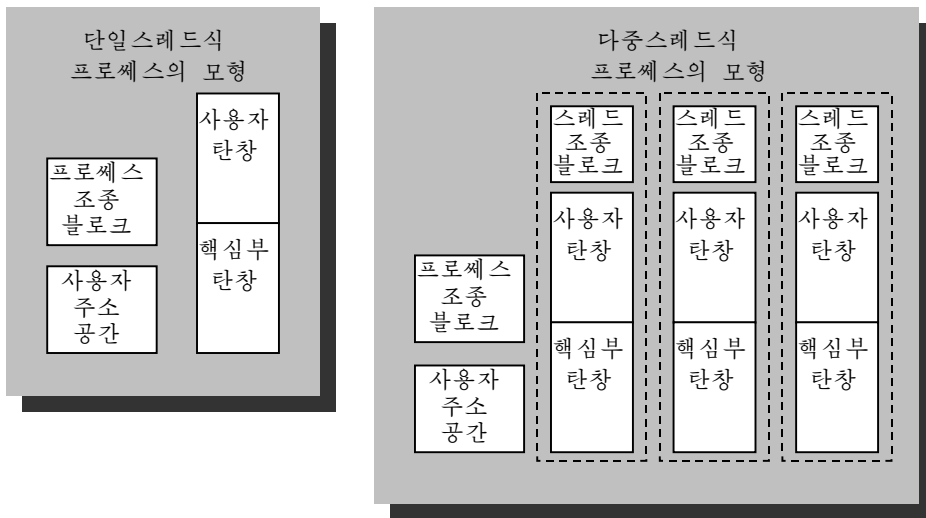



그림 4-2. 단일스레드 및 다중스레드식프로쎄스의 모형

환경에는 여전히 단일프로쎄스조종블로크와 프로쎄스와 사용자주소공간이 있지만 현재는 등록기값, 우선도 그리고 다른 스레드와 관련된 상태정보를 포함하고 있는 매개 스레드용개별조종블로크는 물론 매개 스레드용개별탄창들이 있다.

결과 프로쎄스의 모든 스레드는 그 프로쎄스의 상태와 자원을 공유한다. 그것들은 같은 주소공간에 상주하며 같은 자료에 접근한다. 한개 스레드가 기억기안의 자료항목을 변경시킬 때 다른 스레드는 그것들이 항목에 접근할 때 그 결과를 본다. 만일 한개 스레드가 읽기특권을 가진 파일을 열면 같은 프로쎄스에 있는 다른 스레드도 역시 그 파일에서 읽을수 있다.

스레드의 주요리익은 성능관계로부터 얻어 진다. 즉

1. 어떤 명칭을 가지는 새로운 프로쎄스를 창조하는것보다 현존 프로쎄스안에서 새로운 스레드를 창조하는데 걸리는 시간이 훨씬 작다. Mach개발자들이 진행한 연구는 스레드를 사용하지 않는 비교할만한 UNIX실현물에 비해 프로쎄스창조속도 상승이 몇십배라는것을 보여 주고 있다[TEVA 87].
2. 프로쎄스보다 스레드를 결속하는데 적은 시간이 걸린다.
3. 같은 프로쎄스안에서 두개의 스레드사이를 절환하는데 적은 시간이 걸린다.
4. 스레드는 각이한 집행프로그람들사이 통신에서 효과성을 높여 준다. 대부분의 조작체계에서 독립적인 프로쎄스들사이의 통신은 핵심부가 간섭하여 통신에 필요한 보호와 기구들을 보장하여야 한다. 그러나 같은 프로쎄스의 스레드는 기억기와 파일을 공유하기때문에 핵심부가 없이도 서로 통신할수 있다.

그러므로 집행과 관련되는 단위들의 모임으로서 실현될 응용프로그람이나 기능이 있다면 개별프로쎄스의 집합보다 오히려 스레드를 집합하여 하는것이 훨씬 더 효과적이다.

스레드를 사용할수 있는 응용프로그람의 한 실례는 파일봉사기이다. 새로운 파일요청이 들어 오면 파일관리프로그람을 위해 새로운 스레드를 새끼칠수 있다. 봉사기가 많은 요청을 처리하므로 많은 스레드가 짧은 주기동안에 창조되고 파괴될것이다. 만일 봉

사기가 다중처리기기계에서 실행한다면 같은 프로쎄스의 다중스레드를 동시에 서로 다른 처리기상에서 집행할수 있다. 더우기 파일봉사기안의 프로쎄스나 스레드가 파일자료를 공유해야 하며 그것들의 동작을 조정해야 하기때문에 스레드와 공유기억기를 사용하는것이 조종을 위해 프로쎄스와 통보문넘기기를 사용하는것보다 더 빠르다.

스레드구조는 또한 단일처리기가 몇가지 서로 다른 기능을 론리적으로 수행하는 프로그람구조를 간단화하는데 유용하다.

[LETW88]은 단일사용자다중처리체계에서 스레드사용에 대한 네가지 실례를 주었다. 즉

- **전경 및 배경작업 :** 실례로 자료표프로그람에서 한개 스레드는 차림표를 표시하고 사용자입력을 읽는 동안 다른 스레드는 사용자지령을 집행하고 자료표를 갱신한다. 이 배렬은 흔히 프로그람이 이전 지령을 끝내기전에 다음지령을 재촉하도록 함으로써 응용프로그람의 파악속도를 높여 준다.
- **비동기처리 :** 프로그람에서 비동기요소들을 스레드로 실현할수 있다. 실례로 전원고장보호를 위해 매 분에 한번씩 자유기억완충기의 내용을 디스크에 써넣도록 단어처리기를 설계할수 있다. 단독일감을 주기적으로 예비보관시키며 조작체계로 그자체를 직접 일정작성하는 스레드를 창조할수 있는데 주프로그람의 많은 코드가 시간검사를 하거나 입력 및 출력을 조종할 필요가 전혀 없다.
- **고속집행 :** 다중스레드식프로쎄스는 장치로부터 다음자료의 묶음을 읽는 동안 하나의 자료묶음을 계산할수 있다. 다중처리기체계에서 같은 프로쎄스로부터 여러 스레드를 동시에 집행할수 있다.
- **모듈식프로그람구조 :** 다양한 동작이나 다양한 입출력원천지 및 목적지를 가지는 프로그람들은 스레드를 사용하여 더 쉽게 설계하고 실현할수 있다.

일정작성 및 할당은 스레드에 기초하여 수행되는데 이로부터 집행을 취급하는 대부분의 상태정보는 스레드-수준자료구조에 보존된다. 그러나 프로쎄스에서 모든 스레드에 영향을 주며 또한 조작체계가 프로쎄스준위에서 관리하여야 하는 몇가지 동작이 있다. 중단은 주기억기밖으로 주소공간을 교체하는 과정을 포함한다 .프로쎄스안의 모든 스레드가 같은 주소공간을 공유하기때문에 모든 스레드는 동시에 중단상태에 들어 가야 한다. 마찬가지로 프로쎄스의 결속은 프로쎄스의 모든 스레드를 결속한다.

스레드의 기능성

프로쎄스와 같이 스레드는 집행상태를 가지며 서로 동기시킬수 있다. 이번에는 스레드기능성의 두가지 측면을 본다.

스레드의 상태

프로쎄스와 같이 스레드에서의 기본상태는 실행, 준비 및 폐색이다. 일반적으로 중단상태를 스레드와 관련시키는것은 맞지 않는다. 왜냐하면 그런 상태들은 프로쎄스-준위에 대한 개념들이기때문이다. 특히 프로쎄스가 교체되여 나간다면 그의 모든 스레드는 반드시 교체되여 나간다. 그것은 그것들이 모두 프로쎄스의 주소공간을 공유하기때문이다.

스레드상태에서의 변화와 관련한 네가지 기초적인 스레드조작이 있다[ANDE97]. 즉

- **새끼치기 :** 대표적으로 새로운 프로쎄스가 새끼칠 때 프로쎄스에서의 스레드도 역시 새끼친다. 스레드상태에서의 변화와 관련된 4가지 기본스레드조작이 있다. 그후에 프로쎄스의 스레드는 새로운 스레드에서의 명령지시기와 변수를 주면서 같은 프로쎄스에서 또다른 스레드를 새끼칠수 있다. 새로운 스레드에 자체의 등

록기문맥과 탄창공간을 주며 준비대기렬에 놓는다.

- **폐색 :** 스레드가 사건기다림을 요구할 때 그것은 폐색될것이다(그의 사용자등록기, 프로그람계수기, 탄창지시기들을 보관한다.). 처리기는 이제부터 다른 준비상태의 스레드집행에로 전환할수 있다.
- **비폐색 :** 스레드가 폐색되는 사건이 발생할 때 스레드는 준비대기렬로 이동한다.
- **마무리 :** 스레드가 완료할 때 등록기문맥과 탄창은 재배정된다.

중요한 문제는 스레드의 폐색이 전체 프로쎄스의 폐색을 초래하는가 안하는가 하는것이다. 다른 말로 만일 프로쎄스안의 한개 스레드가 폐색되면 이것이 다른 스레드가 준비상태에 있다고 하더라도 같은 프로쎄스안의 그 어떤 다른 스레드실행을 못하게 하는가? 명백히 한개의 폐색된 스레드가 전체 프로쎄스를 폐색시킨다면 스레드의 일부 유연성과 능력이 상실된다.

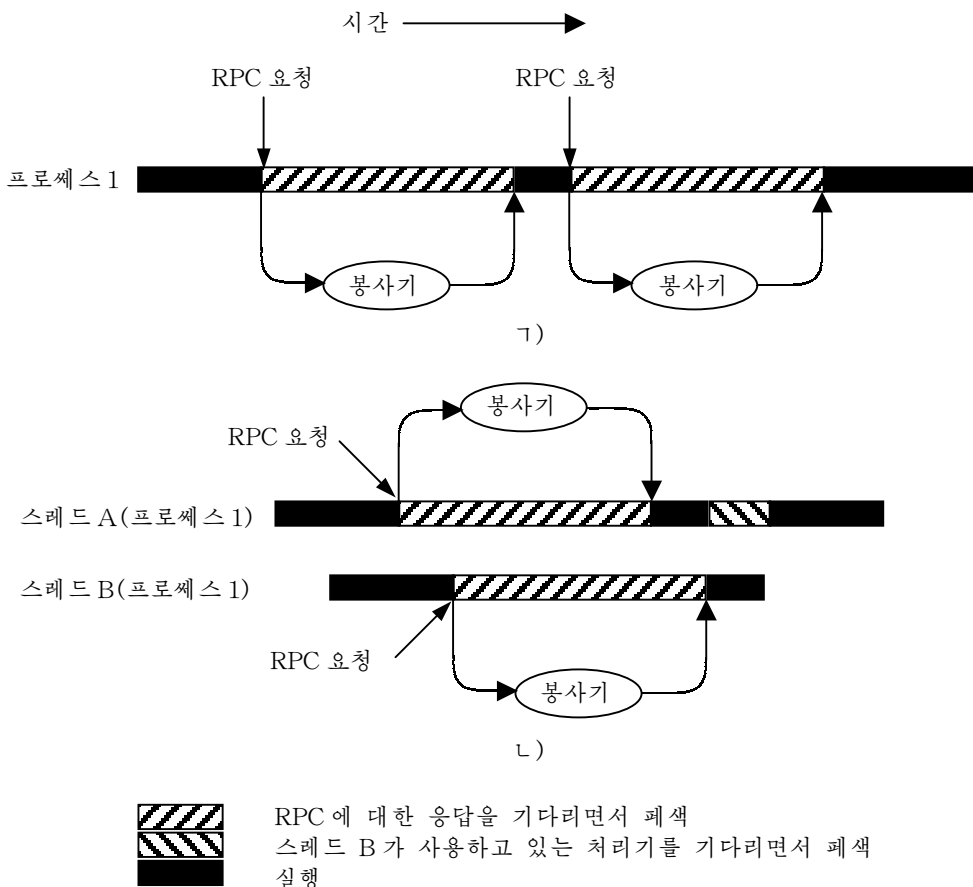



그림 4-3. 스레드를 사용하는 원격수속호출(RPC) : ㄱ-단일스레드를 사용하는 RPC, ㄴ-봉사기당 한개의 스레드를 사용하는 RPC(단일처리기상에서)

핵심부-준위스레드에 대한 사용자-준위스레드의 론의에서 후에 이 문제에 돌아 가지만 현재는 전체 프로쎄스를 폐색시키지 않는 스레드의 성능리익을 고찰하도록 하자. 그림 4-3([KLEI96]의 하나에 기초함)은 결합된 결과를 얻기 위해 두개의 서로 다른 주콤퓨터에 대한 두개의 원격수속호출(RPC) [2]을 수행하는 프로그람을 보여 준다. 단일스레드식프로그람안에서는 결과들이 차례로 얻어 지며 그래서 프로그람은 차례로 매개 봉사기로부터의 응답을 기다려야 한다. 매개 RPC에서 개별스레드를 사용하는 프로그람을 다시 작성하면 속도를 근본적으로 높일수 있다. 만일 이 프로그람을 단일처리기상에서 조작한다고 해도 요청이 련속적으로 발생하므로 결과적으로는 순차적으로 처리되지만 프로그람은 두개의 응답을 병행하여 기다린다.

단일처리기상에서 다중프로그람처리는 다중프로쎄스에서 다중스레드를 교차처리할수 있게 한다. 그림 4-4의 실례에서는 두개의 프로쎄스에서 세개의 스레드가 처리기상에서 교차처리된다. 현재 실행중인 스레드가 폐색되거나 그의 시간쪼각을 다 써버리면 집행은 한 스레드에서 다른 스레드에로 넘어 간다.[3]

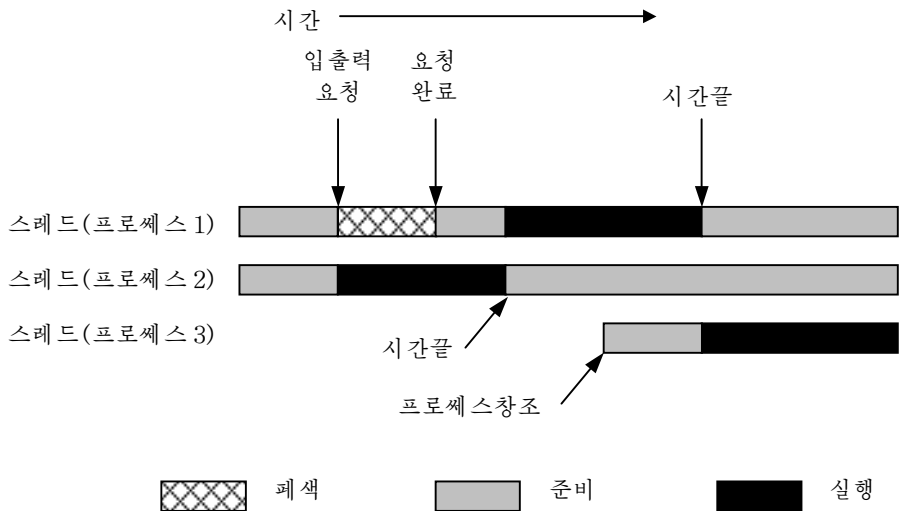



그림 4-4. 단일처리기상에서의 다중스레드처리실례

---

[2] RPC는 서로 다른 기계상에서 집행할수 있는 두 프로그람이 수속호출/복귀의 문법과 의미론을 사용하여 대화하는 수법이다. 호출 및 피호출프로그람들은 량자가 마치 협조프로그람이 같은 기계상에서 실행하듯이 동작한다. RPC는 흔히 의뢰기/봉사기응용프로그람들에서 쓰이는데 제13장에서 설명한다.

[3] 이 실례에서 스레드 C는 스레드 B가 실행할 준비가 되여 있어도 스레드 A가 그의 시간을 다 써버린 다음에 실행하기 시작한다. B인가 C인가 하는 선택은 일정작성결심이며 이 문제는 제4편에서 고찰한다.

스레드의 동기화

프로쎄스의 모든 스레드는 파일열기와 같은 주소공간과 다른 자원을 공유한다. 한개 스레드가 자원을 임의로 변경시키는것은 같은 프로쎄스안에서 다른 스레드의 환경에 영향을 준다. 그러므로 스레드들이 서로 간섭하거나 자료구조를 파괴시키지 않도록 여러가지 스레드의 동작을 동기화해야 한다. 실례로 만일 두개의 스레드가 각각 어떤 요소를 2중적으로 련결된 목록에 추가하려고 한다면 한개 요소를 잃을수도 있고 또는 목록이 파괴될수도 있다.

스레드의 동기화에서 생기는 문제와 사용된 수법은 일반적으로 프로쎄스의 동기화에서와 꼭같다. 이 문제와 수법은 제5장 및 제6장의 주제이다.

실례 Adobe PageMaker

스레드사용실례로서는 Linux와 같은 공유체계하에서 실행하는 Adobe PageMaker의 응용프로그람이다. Pagemaker는 탁상출판에서의 편집, 설계 및 제작도구이다. 그림 4-5[KRON90]에서 보여 준 스레드구조는 응용프로그람의 응답성을 최량화하도록 선택한것이다. 세개의 스레드(즉 사건처리스레드, 화면재작도스레드 및 봉사스레드)는 항상 능동적이다.

일반적으로 OS/2는 입구통보가 너무 많은 처리를 요구한다면 창문관리에서의 응답을 적게 한다. OS/2는 어떤 통보도 0.1s이상의 처리시간을 요구하지 않는 상태로 안내한다. 실례로 인쇄지령을 처리하는 동안 페지를 인쇄하기 위하여 보조루틴을 호출하는것은 성능을 낮추면서 어떤 통보를 더이상 다른 응용프로그람들에 할당하지 않도록 체계를 보호한다. 이 기준을 만족시키기 위해 PageMaker에서 시간을 소비하는 사용자조작 즉 자료인쇄 및 자료수입 그리고 본문흘리기들은 봉사스레드가 수행한다. 프로그람의 초기화도 크게는 봉사스레드가 수행하는데 그 스레드는 사용자가 새로운 문서를 창조하거나 현존문서를 열기 위한 대화를 기동시키는 동안의 휴식시간을 흡수한다. 개별스레드는 새로운 사건통보문을 기다린다.

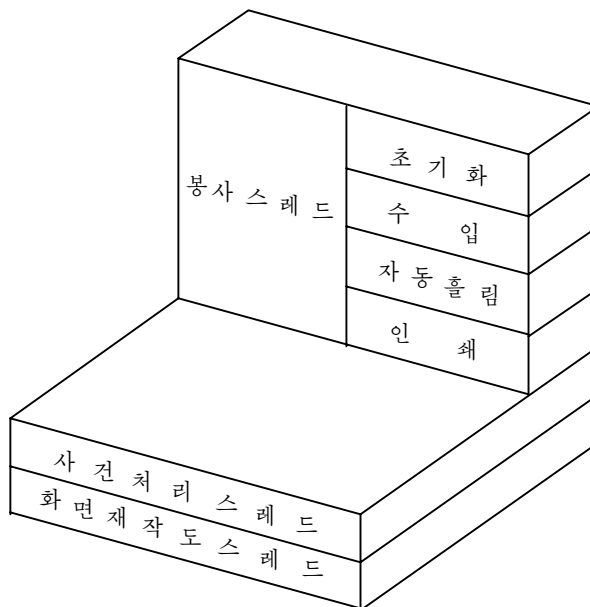



그림 4-5. Adobe PageMaker에서의 스레드구조

봉사스레드 및 사건처리스레드를 동기시키는것은 복잡하다. 왜냐하면 봉사스레드가 여전히 경쟁하고 있을 때 사용자가 사건처리스레드를 활성화시키는 마우스의 누르기나 이동을 계속할수 있기때문이다. 만일 이런 충돌이 발생하면 PageMaker는 이 통보문들을 려파하고 창문크기지정과 같은 일정한 기초적인것들만 접수한다.

통보문은 봉사스레드로부터 전달되여 그의 과제가 완료되였다는것을 가리킨다. 이것이 발생할 때까지 PageMaker에서의 사용자활동성은 제한된다. 프로그람은 차림표항목을 없애고 《동작중》 유표를 표시하여 이것을 가리킨다. 사용자가 다른 응용프로그람에로 자유롭게 절환하여 동작중 유표가 다른 창문으로 움직일 때 그것은 응용프로그람에서 알맞는 유표로 변화된다.

개별스레드는 두가지 리유로 화면재작도에 사용된다. 즉

1. PageMaker는 몇개의 객체가 한 페지에 나타나는것을 제한하지 않는다. 결과 재작도요청의 처리가 0.1s의 안내시간을 쉽게 초과할수 있다.
2. 개별스레드를 사용하면 사용자가 작도를 류산시키게 할수 있다. 이 경우 사용자가 페지에 척도를 다시 정할 때 즉시에 재작도를 계속할수 있다. 낡은 척도에서의 페지현시를 마무리하고 그다음 새로운 척도에서 재작도한다면 프로그람은 더 적게 응답한다.

동적흘리기사용자가 흘리기표식자를 끌어 당기는데 따라 화면을 재작도할수도 있다. 사건처리스레드는 흘림띠를 감시하고 여백눈금을 재작도한다(그것은 빨리 재작도하며 즉시에 사용자에게 위치를 귀환시켜 준다.). 한편 화면재작도스레드는 항시적으로 페지를 재작도하며 따라 잡으려고 한다.

여러개의 스레드를 사용하지 않고 동적재작도를 수행하는것은 많은 점에서 통보문을 등록하는 응용프로그람에 더 큰 부담을 준다. 다중스레드처리는 병행동작들이 코드상에서 더 자연스럽게 분리되게 한다.

사용자준위 및 핵심부준위스레드

스레드실현에 대한 두가지의 넓은 범주가 있다. 즉 사용자준위스레드(ULT)와 핵심부준위스레드(KLT)이다.[4] 후자는 또한 핵심부가 지원하는 스레드 또는 경량프로쎄스로 문헌들에 언급되여 있다.

사용자준위스레드

순수한 ULT기능에서 모든 스레드관리작업은 응용프로그람에 의해 수행되며 핵심부는 스레드의 존재를 알지 못한다. 그림 4-6 ㄱ는 순수한 ULT법을 서술한다. 임의의 응용프로그람은 ULT관리를 위한 루틴제품인 스레드서고를 사용함으로써 프로그람을 다중스레드화되도록 작성할수 있다. 스레드서고는 스레드창조 및 소거, 스레드사이에서 통보문 및 자료의 넘기기, 스레드집행의 일정작성 그리고 스레드문맥보관 및 회복을 위한 코드를 가진다.

기정사실로부터 응용프로그람은 하나의 스레드로 시작하며 그 스레드에서 실행할 때 시작한다. 응용프로그람과 스레드는 핵심부가 관리하는 단일프로쎄스에 배정된다. 응용프로그람은 실행중 임의의 시각에(프로쎄스는 실행중 상태에 있다.) 새로운 스레드를 새끼쳐서 같은 프로쎄스안에서 실행시킬수 있다. 새끼치기는 스레드서고에서 새끼치기편의프로그람을 기동하여 수행한다. 조종은 수속호출에 의해 편의프로그람에 넘어 간다. 스레드서고는 새로운 스레드에서의 자료구조를 참조하며 준비상태에 있는 프로쎄스안의 한 스

---

4. 략어 ULT와 KLT는 이 책에서 처음으로 간결하게 하기 위하여 받아 들였다.

레드에 조종을 넘긴다. 조종이 서고에 넘어갈 때 현재스레드의 문맥은 보관되며 조종이 서고로부터 스레드에로 넘어 갈 때 스레드의 문맥은 회복된다. 문맥은 본질적으로 사용자등록기, 프로그람계수기, 그리고 탄창지시기의 내용들로 구성되여 있다.

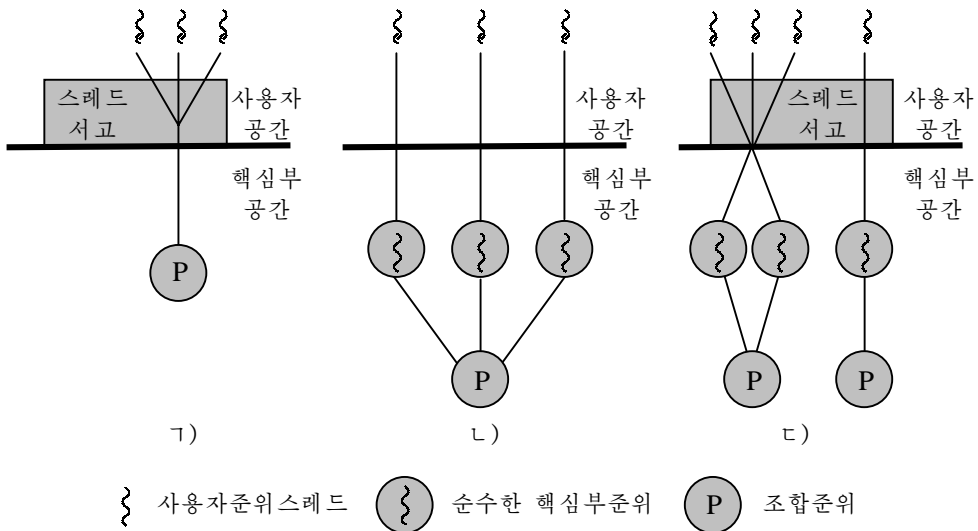



그림 4-6. 리용자준위 및 핵심부준위스레드
ㄱ-순수한 사용자준위, ㄴ-순수한 핵심부준위, ㄷ-조합준위

앞의 단락에서 서술한 모든 동작은 사용자공간에서 그리고 단일프로쎄스안에서 일어난다. 핵심부는 이 동작을 알아채지 못한다. 핵심부는 하나의 단위로서 프로쎄스를 계속 일정작성하며 프로쎄스에 하나의 집행상태(준비, 실행, 페색 등)를 할당한다. 다음의 실례는 스레드일정작성과 프로쎄스일정작성사이의 관계를 명백히 해준다. 프로쎄스 B가 스레드 2에서 집행중에 있다고 가정하자. 프로쎄스의 상태와 프로쎄스에 속하는 두개의 ULT를 그림 4-7 ㄱ에서 보여 주었다. 다음의것들중에서 매개가 일어 날수 있는것들이다.

1. 스레드 2에서 집행중인 응용프로그람이 B를 페색시키는 체계호출을 한다. 실례로 입출력호출이 이루어 진다. 이것은 조종이 핵심부에 넘어 가도록 한다. 핵심부는 입출력동작을 기동시키며 프로쎄스 B를 페색상태에 놓고 다른 프로쎄스에로 절환한다. 한편 스레드서고가 가지고 있는 자료구조에 따라 프로쎄스 B의 스레드 2는 여전히 실행중 상태에 있다. 스레드 2가 처리기상에서 집행된다는 의미에서 실지로 실행중이 아니라는것에 주목하는것이 중요하지만 스레드서고에 의해 실행중 상태에 있는것으로 느껴 진다. 해당한 상태도를 그림 4-7 ㄴ에 보여 주었다.
2. 박자새치기가 핵심부에로 조종을 넘기며 핵심부는 현재 실행중인 프로쎄스(B)가 그의 시간소편을 다 소모했다는것을 판정한다. 핵심부는 프로쎄스 B를 준비상태로 놓고 다른 프로쎄스에로 절환한다. 한편 스레드서고가 가지고 있는 자료구조에 따라 프로쎄스 B의 스레드 2는 여전히 실행중 상태에 있다. 해당한 상태도를 그림 4-7 ㄷ에 보여 주었다.
3. 스레드 2가 프로쎄스 B의 스레드 1이 수행하는 일부 동작을 요구하는 점에 이르렀다. 스레드 2는 페색상태에 들어 가고 스레드 1은 준비로부터 실행중으로 이행한다. 프로쎄스 그자체는 실행상태에 남아 있는다. 해당한 상태도를 그림 4-7 ㄹ에 보여 주었다.

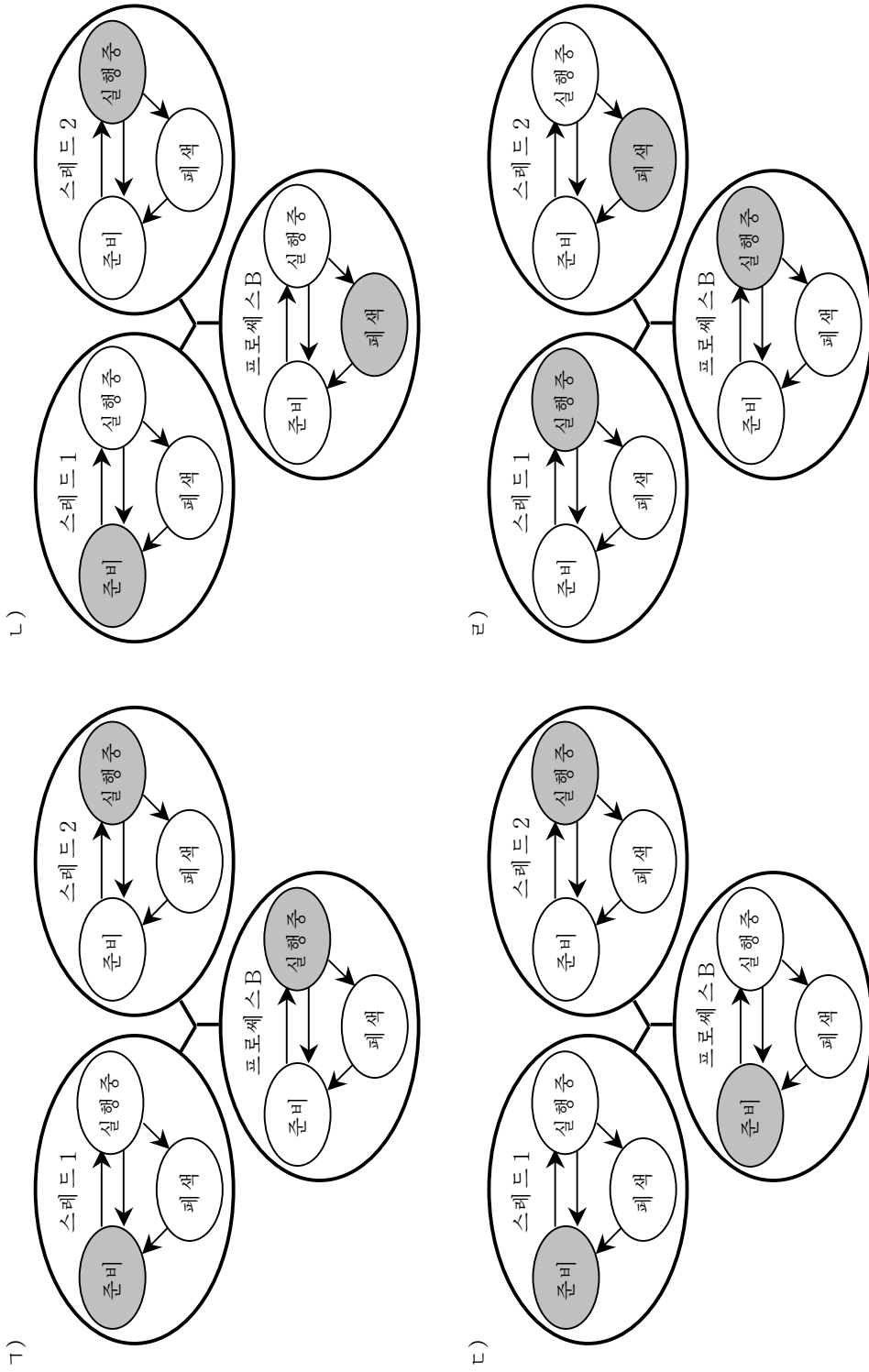



그림 4-7. 사용자준위스레드상태와 프로쎄스상태사이의 관계실례

1과 2의 경우에 (그림 4-7 ㄴ와 4-7 ㄷ) 핵심부가 프로쎄스 B에로 조종을 거꾸로 절환하면 집행은 스레드에서 계속한다. 스레드서고의 코드를 집행하는 동안 프로쎄스가 그의 시간소편을 다 소모하거나 높은 우선권프로쎄스가 선취되여 새치기될수 있다는것을 주목하자. 결국 프로쎄스는 새치기될 때 스레드로부터 다른 스레드에로의 스레드절환의 중간에 있을수 있다. 프로쎄스가 다시 시작할 때 집행은 스레드서고에서 계속되는데 프로쎄스는 스레드절환을 끝내고 프로쎄스안에 있는 새로운 스레드에 조종을 이송한다.

KLT대신에 ULT를 사용하면 다음과 같은 몇가지 우점이 있다. 즉

1. 스레드절환은 모든 스레드관리자료구조가 있는 단일프로쎄스의 사용자주소공간에 있기때문에 핵심부방식특권을 요구하지 않는다. 그러므로 프로쎄스는 스레드관리를 하기 위해 핵심부방식에로 절환하지 않는다. 이것은 두 방식절환의 간접소비시간을 절약한다(사용자방식으로부터 핵심부방식에로; 핵심부방식으로부터 다시 사용자방식에로).
2. 일정작성은 응용프로그람에 따라 특징이 있을수 있다. 한 응용프로그람이 단순한 순환일정작성으로부터 리익을 얻을수 있다면 한편 다른 응용프로그람은 우선권에 기초한 일정작성알고리듬으로부터 리익을 얻을수 있다. 기초로 되여 있는 OS일정작성기를 방해하지 않고 응용프로그람에 일정작성알고리듬을 주문할수 있다.
3. ULT는 임의의 조작체계상에서 실행할수 있다. 기초로 되여 있는 핵심부가 ULT를 지원하는데서 아무런 변화도 요구하지 않는다. 스레드서고는 모든 응용프로그람이 공유하는 응용프로그람준위편의프로그람의 모임이다.

KLT에 비해 ULT에는 두가지 명백한 결함이 있다. 즉

1. 대표적인 조작체계에서 많은 체계호출들은 페색중에 있다. 결과 ULT가 체계호출을 집행할 때 스레드가 페색될뿐아니라 프로쎄스의 모든 스레드가 페색된다.
2. 순수한 ULT전략에서 다중스레드식응용프로그람은 다중처리의 우점을 가질수 없다. 핵심부는 한번에 오직 한개의 처리기에 한개의 프로쎄스를 할당한다. 그러므로 프로쎄스의 단일한 스레드만이 한번에 집행할수 있다. 실제상 단일프로쎄스에 응용프로그람준위다중프로그람처리가 있다. 다중프로그람처리가 응용프로그람의 속도를 크게 높여 주는 한편 코드부분을 병행하여 집행하는 능력으로 하여 리익을 주는 응용프로그람들이 있다.

이 두가지 문제에 대한 작업방법들이 있다. 실례로 두 문제는 다중스레드보다 오히려 다중프로쎄스로 응용프로그람을 작성하여 극복할수 있다. 그러나 이 방법은 스레드의 주요우점을 없앤다. 즉 매개 절환은 스레드절환이 아니라 프로쎄스절환으로 되여 더 큰 간접소비시간을 가지게 된다.

페색중인 스레드문제를 극복하는 다른 방법은 쟈케트화수법을 사용하는것이다. 쟈케트화의 목적은 페색중인 체계호출을 비페색중인 체계호출로 변환하는것이다. 실례로 체계입출력루틴을 직접호출하는 대신에 스레드가 응용프로그람준위의 입출력쟈케트루틴을 호출한다. 쟈케트루틴에서 입출력장치가 동작중인가를 판정하기 위하여 검사하는 코드가 있다. 만일 그렇다면 스레드는 준비상태에 들어 가고 조종을 다른 스레드에 넘긴다(스레드서고를 통하여). 이 스레드가 후에 다시 조종을 받을 때 그것은 다시 입출력장치를 검사한다.

핵심부준위스레드

순수한 KLT기능에서 모든 스레드의 관리작업은 핵심부가 수행한다. 응용프로그람령역에는 간단히 핵심부스레드기능에 대한 응용프로그람대면부(API)가 있고 스레드관리코

드는 전혀 없다. W2K, Linux 그리고 OS/2가 이 방법에 대한 실례로 된다.

그림 4-6 ㄴ는 순수한 KLT방법을 보여 주고 있다. 임의의 응용프로그람을 다중스레드식으로 작성할수 있다. 응용프로그람의 모든 스레드는 단일프로쎄스에서 지원을 받는다. 핵심부는 총적으로 프로쎄스를 위한 그리고 그 프로쎄스의 개별적인 스레드들을 위한 문맥정보를 가지고 있다. 핵심부에 의한 일정작성은 스레드를 기초로 하여 수행된다. 이 방법은 ULT방법의 두가지 원리적약점을 극복한다. 첫째로, 핵심부는 다중처리기상에서 같은 프로쎄스로부터 다른 스레드를 동시에 일정작성할수 있다. 둘째로, 프로쎄스안의 한개 스레드가 페색되면 핵심부는 같은 프로쎄스의 다른 스레드를 일정작성할수 있다. KLT방법의 다른 우점은 핵심부루틴 그자체를 다중스레드화할수 있다는것이다.

ULT방법에 비한 KLT방법의 원리적인 결함은 같은 프로쎄스에서 한 스레드로부터 다른 스레드에로 조종을 넘길 때 핵심부에 방식절환을 요구한다는것이다. 이 차이를 설명하기 위해 표 4-1은 UNIX와 같은 조작체계를 실행하는 단일처리기 VAX기계에서 얻은 결과를 보여 준다. 두개의 기준점은 다음과 같다. 즉 빈 갈래는 령수속(즉 프로쎄스/스레드를 갈라 주는데서의 간접소비시간)을 기동하는 프로쎄스/스레드를 창조, 일정작성, 집행 및 완료하는데 걸리는 시간이며 그리고 신호-기다림은 프로쎄스/스레드가 기다리고 있는 프로쎄스/스레드에 신호를 준 다음 조건(즉 두 프로쎄스/스레드를 함께 동기화시키는 간접소비시간)을 기다리는 시간이다. 이 수들을 원근화법에 넣으면 이 연구에서 사용된 VAX상에서의 수속호출은 약 7㎲ 걸리고 한편 핵심부내부새치기는 약 17㎲ 걸린다. ULT와 KLT사이에는 크기순서에서 더 큰 차이가 있으며 KLT와 프로쎄스사이는 비슷하다.

결과 그에 대해서 보면 단일스레드식프로쎄스에 비해 KLT다중스레드처리를 사용함으로써 중요하게 속도가 제고되는 한편 ULT를 사용하여 추가적으로 중요하게 속도가 제고된다. 그러나 추가적인 속도제고가 실현되는가 안되는가 하는것은 포함된 응용프로그람의 특징에 관계된다. 응용프로그람에서 대부분의 스레드절환이 핵심부방식접근을 요구하면 ULT에 기초한 방안은 KLT에 기초한 방안보다 성능이 훨씬 더 좋지 못할수 있다.

표 4-1. 스레드의 조작지연(㎲)[ANDE 92]

조작	사용자준위스레드	핵심부준위스레드	프로쎄스
빈 갈래	34	948	11300
신호기다림	37	441	1840

결합방법

일부 조작체계는 결합된 ULT/KLT기능을 준다(그림 4.6 ㄷ). Solaris가 이에 대한 기본적인 실례로 된다. 결합체계에서 스레드창조는 응용프로그람에서 스레드일정작성과 동기화의 범위이므로 사용자공간에서 완전히 수행된다. 단일응용프로그람의 다중 ULT는 일정한(작거나 같은) 수의 KLT상에 사영된다. 프로그람작성자는 가장 좋은 전체적인 결과를 얻기 위해 특정한 응용프로그람과 기계에서의 KLT의 수를 조절할수 있다.

결합방법에서 같은 응용프로그람의 여러개 스레드는 여러 처리기상에서 병렬로 실행할수 있고 페색중인 체계호출요구가 전체 프로쎄스를 페색시키지 않는다. 적절히 설계한다면 이 방법은 다른 배렬결합을 최소로 되게 하는 한편 순수한 ULT와 KLT방법의 우점을 결합시킬것이다.

표 4-2. 스레드와 프로쎄스사이의 관계

스레드 : 프로쎄스	서 술	실례체계
1:1	집행중의 매개 스레드는 자체의 주소공간과 자원을 가진 단일프로쎄스이다.	전통적인 UNIX실현물
M:1	프로쎄스는 주소공간 및 동적자원소유권을 정의한다. 다중스레드는 프로쎄스에서 창조되고 집행된다.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	스레드는 한 프로쎄스환경으로부터 다른 프로쎄스환경에로 이주할수 있다. 이것은 별개의 체계사이에서 스레드가 쉽게 이동하도록 한다.	Ra(Clouds), Emerald
M:N	M:1과 1:M경우의 속성들을 결합시킨다.	TRIX

다른 배렬

앞에서 말한바와 같이 자원배정 및 할당에 대한 개념은 전통적으로 단일한 프로쎄스의 개념에 포함되여 왔었다. 즉 스레드와 프로쎄스사이의 관계는 1대 1이다. 최근에 다수대 1관계인 단일프로쎄스에서 여러 스레드를 주는데 많은 주목이 돌려 졌다. 그러나 표 4-2에서 보는바와 같이 다른 두가지 결합 이른바 다수대 다수관계 및 1대 다수관계를 연구하였다.

다수대 다수관계

스레드와 프로쎄스사이에서 다수대 다수관계를 가지는 문제는 실험적인 조작체계 TRIX[SIEB83, WARD80]에서 개발되였다. TRIX에는 령역과 스레드의 개념이 있다. 령역은 통보문을 보내고 받을수 있는 주소공간 및 《포구》로 이루어 진 정적인 실체이다. 스레드는 집행탄창, 처리기의 상태 그리고 일정작성정보를 가진 단일집행경로이다.

지금까지 다중스레드처리방법을 론의한바와 같이 여러 스레드는 초기에 론의된 효과성리득을 주면서 단일령역에서 집행할수 있다. 그러나 단일사용자동작이나 응용프로그람도 또한 다중령역에서 동작할수 있다. 이 경우에 한 령역으로부터 다른 령역에로 이동할수 있는 스레드가 존재한다.

다중령역에서 단일스레드를 사용하는것은 초기에 프로그람작성자에게 구조화도구를 주려는데로부터 나온것 같다. 실례로 입출력부분프로그람을 사용하는 어떤 프로그람을 고찰하자. 사용자가 새끼친 프로쎄스를 처리하는 다중프로그람처리환경에서 주프로그람이 입출력을 처리하고 다음집행을 계속하기 위해 새로운 프로쎄스를 창조할수 있다. 그러나 주프로그람의 앞으로의 진척이 입출력조작의 결과에 의존한다면 주프로그람은 다른 입출력프로그람이 끝날 때까지 기다려야 할것이다. 이 응용프로그람을 완성하는 몇가지 방법이 있다.

1. 전체프로그람을 단일프로쎄스처럼 실현할수 있다. 이것은 합리적이며 직선적인 풀이이다. 기억기관리와 관련되는 약점이 있다. 총적으로 프로쎄스가 효과적으로 집행하기 위해 상당한 주기억기를 요구할수 있는 반면에 입출력을 완충시키고 상대적으로 작은 프로그람코드를 처리하는데서 상대적으로 작은 주소공간을 요구한다. 입출력프로그람이 규모가 더 큰 프로그람의 주소공간에서 집행하기때

문에 전체적인 프로쎄스가 입출력조작기간 주기억기에 남아 있든가 그렇지 않으면 입출력조작이 교체되든가 해야 한다. 이 기억기관리의 효과는 또한 주프로그람과 입출력부분프로그람이 같은 주소공간에서 두개의 스레드로서 실현되는 경우에 있게 된다.

2. 주프로그람과 입출력부분프로그람을 두개의 개별프로쎄스로 실현할수 있다. 이것은 종속적인 프로쎄스창조로 인한 간접소비시간을 초래한다. 만일 입출력동작이 빈번하다면 관리자원을 소비하는 종속적인 프로쎄스도 살려 놓아야 한다. 그렇지 않으면 비효과적인 부분프로그람을 자주 창조하거나 파괴한다.
3. 단일스레드로 실현되는 단일동작으로서 주프로그람과 입출력보조프로그람을 취급하자. 그러나 한개 주소공간(령역)은 주프로그람용으로 창조하고 다른것은 입출력부분프로그람용으로 창조할수 있다. 결과 스레드는 집행절차에 따라 두개의 주소공간사이에서 이동할수 있다. 조작체계는 독립적으로 두개의 주소공간을 관리할수 있으므로 그 어떤 프로쎄스창조의 간접소비시간도 생기지 않는다. 게다가 입출력부분프로그람에 사용된 주소공간은 다른 간단한 입출력프로그람이 공유할수도 있다.

TRIX개발자들의 경험은 세번째 선택이 우점을 가지고 있으며 일부 응용프로그람에서 가장 효과적인 풀이로 된다는것을 보여 주고 있다.

1대 다수관계

분산조작체계분야(분산콤퓨터체계를 조종하기 위해 설계된)에서는 주소공간사이에서 이동할수 있는 최초의 실체로서 스레드의 개념에 흥미를 가지였다.[5] 이 연구에서 주시할만한 실례로서는 Clouds조작체계를 들수 있으며 특히 Ra[DASG 92]라고 하는 그의 핵심부를 들수 있다. 다른 실례로서는 Emerald체계[STEE 95]를 들수 있다.

Clouds에서 스레드는 사용자의 원근화법으로부터 본 동작의 단위이다. 프로쎄스는 련관된 프로쎄스조종블로크를 가진 가상주소공간이다. 창조된후 스레드는 프로쎄스안의 프로그람에 대해 입구점을 기동시켜 프로쎄스안에서 집행을 시작한다. 스레드는 한개 주소공간으로부터 다른 곳으로 이동할수 있으며 실지로는 기계들을 경계로 움직인다(즉 한 콤퓨터로부터 다른 콤퓨터에로 이동한다.). 스레드가 움직일 때 그것은 말단조종, 전역변수 그리고 일정작성안내(실례로 우선권)와 같은 일정한 정보를 그와 함께 날라야 한다.

Clouds의 방법은 사용자와 프로그람작성자를 분산환경의 세부로부터 격리시키는 효과적인 방법을 준다. 사용자의 동작을 단일스레드로 표현할수 있으며 원격자원에 접근할데 대한 요구 및 적재균형과 같은 여러가지 체계와 관련된 리유로부터 조작체계가 그 기계들사이에서 스레드의 이동을 명령할수 있다.

제 2 절. 대칭형다중처리

전통적으로 콤퓨터를 순서기계로 보아 왔다. 대부분의 콤퓨터프로그람작성언어는 프로그람작성자가 명령의 순차로서 알고리듬을 지정해 주어야 한다. 처리기는 어떤 순차에 따라 한번에 하나씩 기계명령을 집행하여 프로그람을 집행한다. 매개 명령은 연산의 순서대로 집행된다(불러내기명령, 불러내기연산수, 연산의 수행).

---

5. 주소공간사이에서 프로쎄스나 스레드의 이동, 서로 다른 기계들상에서 스레드의 이주는 최근년간에 맹렬한 화제로 되고 있다. 이 문제는 제 14 장에서 고찰한다.

콤퓨터에 대한 이런 견해가 전적으로 옳은것은 결코 아니다.마이크로연산준위에서는 다중조종신호들이 동시에 발생된다. 명령의 관흐름화, 적어도 연산의 불러내기 및 집행조작을 겹치게 하기까지는 오랜 시간이 걸렸다. 이것들은 기능을 병렬로 수행하는 실례들이다.

콤퓨터기술이 발전되고 콤퓨터하드웨어의 가격이 떨어 짐에 따라 콤퓨터설계자들은 일반적으로는 특성을 개선하기 위해, 일부 경우에는 믿음성을 개선하기 위하여 더욱더 병렬기구를 탐구하였다. 이 책에서는 처리기들을 여러개 사용하여 즉 대칭형다중처리기(SMP)와 클라스터를 사용하여 병렬기구를 주는 가장 대중적인 두가지 방법을 론의한다. SMP들에 대해서는 이 절에서, 클라스터들에 대해서는 제6편에서 설명한다.

SMP구성방식

SMP구성방식이 병렬처리기들의 총적범주에서 어디에 어울리는가를 알 필요가 있다. Flynn[FLYN 72]에서 처음으로 소개한 병렬처리기체계를 강조하는 분류법은 여전히 그런 체계를 분류하는 가장 일반적인 방법이다. Flynn은 콤퓨터체계에 대한 다음과 같은 범주들을 제기하였다.

- **단일명령단일자료(SISD)흐름 :** 단일처리기가 단일기억기안에 기억된 자료를 연산하기 위해 단일명령흐름을 집행한다.
- **단일명령다중자료(SIMP)흐름 :** 단일기계명령은 고정걸음에 기초하여 여러 처리요소들의 동시적집행을 조종한다. 매개 처리요소는 련관된 자료기억기를 가지며 매개 명령은 각이한 자료묶음에 기초하여 서로 다른 처리기들이 집행한다. 벡토르 및 배렬처리기들이 이 범주에 속한다.
- **다중명령단일자료(MISD)흐름 :** 자료형을 처리기의 모임에 전송하고 매개 처리기가 서로 다른 명령렬을 집행한다. 이 구조는 결코 완성된것이 아니다.
- **다중명령다중자료(MIMD)흐름 :** 처리기의 모임이 동시에 서로 다른 자료묶음상에서 서로 다른 명령렬을 집행한다.

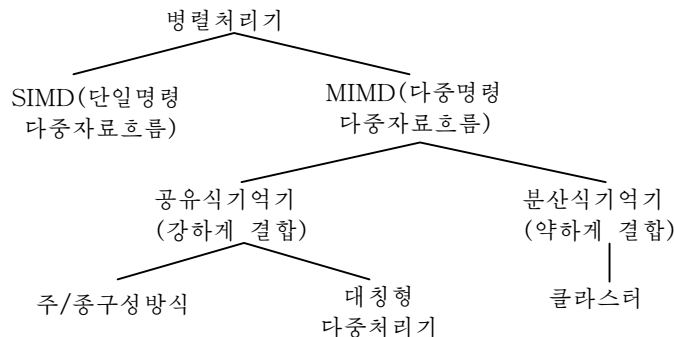



그림 4-8. 병렬처리기방식

MIMD구조를 가지는것이 처리기들의 일반적목적이다. 왜냐하면 그것들이 적당한 자료전송을 수행하는데 필요한 모든 명령을 처리할수 있어야 하기때문이다. MIMD를 처리기가 통신한다는 의미에서 더 세분화할수 있다(그림 4-8). 처리기들이 각각 전용기억기를 가진다면 그때 매개 처리요소는 자체를 포함한 콤퓨터이다. 콤퓨터들사이의 통신은 고정경로를 거치거나 일정한 망기능을 거친다. 그런 체계를 **클라스터** 또는 다중콤퓨터라고 한다. 만일 처리기가 일반기억기를 공유한다면 그때 매개 처리기는 공유기억기안에 기억된 프로그람과 자료에 접근하며 처리기들은 그 기억기를 통하여 서로 통신하는데 그

러한 체계를 **공유기억기다중처리기**라고 한다.

공유기억기식다중처리기에 대한 하나의 일반적인 분류는 프로쎄스들을 처리기에 배정하는 방법에 기초하고 있다. 두가지 기본적인 방법은 주인/종속방법 및 대칭법이다. **주/종속구성방식**을 사용하여 조작체계핵심부는 늘 특정한 처리기상에서 실행한다. 다른 처리기들은 오직 사용자프로그람과 경우에 따라 조작체계편의프로그람들을 집행할수 있다. 주프로쎄스와 스레드의 일정작성을 담당한다. 일단 프로쎄스/스레드가 능동이고 만일 종속처리기가 봉사를 요구한다면(실례로 입출력호출) 그것은 주처리기에 요구를 보내며 봉사가 수행되기를 기다려야 한다. 이 방법은 아주 간단하며 단일처리기식 다중프로그람처리조작체계에서 약간한 개선을 요구한다. 한개의 처리기가 모든 기억기와 입출력자원을 조종하기때문에 충돌해결방도는 간단하다. 이 방법의 결함은 다음과 같다.

- 주처리기의 고장이 체계전체를 허물어 버린다.
- 주처리기는 병목특성을 일으킬수 있다. 왜냐하면 그것이 혼자서 모든 일정작성과 프로쎄스관리를 해야 하기때문이다.

대칭형다중처리기에서 핵심부는 임의의 처리기상에서 집행할수 있으며 대체로 매개 처리기는 사용할수 있는 프로쎄스나 스레드의 모임으로부터 자체일정작성을 한다. 핵심부의 일부가 병렬로 집행하도록 하면서 핵심부를 다중프로쎄스나 다중스레드로 구축할수 있다. SMP방법은 조작체계를 복잡하게 만든다. 두개의 처리기는 결코 같은 프로쎄스를 선택하지 않는다는것과 프로쎄스들이 어떻게 해서든 대기렬에서 상실되지 않는다는것을 담보해야 한다. 자원에 대한 요구를 해결하고 동기화시키기 위한 수법들을 써야 한다.

SMP와 클라스터의 설계는 물리적구성, 호상련결구조, 처리기사이통신, 조작체계설계 및 응용프로그람쏘프트웨어의 수법과 관련되는 문제들을 포함하므로 복잡하다. 여기서 우리의 관심과 후에 클라스터에 대한 설명(제13장)에서 두 경우 다 방식에 대해서 간단히 다치지만 초보적으로는 조작체계설계문제로 된다.

SMP의 조직

그림 4-9는 SMP의 일반적인 조직을 보여 준다. 여러개의 처리기가 있으며 그 매개는 자체조종단, 산수론리단 그리고 등록기를 가지고 있다. 매개 처리기는 일정한 호상련결기구를 통하여 공유된 주기억기와 입출력장치에 접근하는데 공유된 모선은 공통설비로 된다. 처리기는 기억기를 통하여 서로 통신할수 있다(통보문과 상태정보는 공유된 주소공간에 남아 있게 된다.). 처리기가 신호를 직접 교환할수도 있다. 기억기는 흔히 기억기의 개별블로크에 대한 다중동시접근을 할수 있도록 구성된다.

현대 기계들에서 처리기는 일반적으로 그 처리기에 전용인 적어도 한개 준위의 캐쉬를 가진다. 캐쉬를 사용하는것은 일부 새로운 설계문제들을 고찰하게 한다. 매개 캐쉬가 주기억기의 일부분에 대한 상을 포괄하기때문에 만일 한개 단어가 한개 캐쉬에서 변경되면 그것은 다른 캐쉬안에 있는 어떤 단어가 유효성을 상실하게 할수 있다. 이를 막기 위해 다른 처리기들에 갱신이 일어 난다는것을 경보해야 한다. 이 문제를 캐쉬간섭문제라고 하며 대체로 조작체계로서가 아니라 하드웨어로 설명한다.[6]

다중처리기조작체계설계에서 고려할 점

SMP조작체계는 사용자가 체계를 다중프로그람처리식단일처리기체계와 같은 방식으로 볼수 있도록 처리기와 다른 콤퓨터자원들을 관리한다. 사용자는 단일처리기를 사용하

6. 하드웨어에 기초한 캐쉬간섭방안에 대한 설명은 [STAL 00]에서 주고 있다.

겠는지 다른 처리기를 사용하겠는지는 고려하지 않고 프로쎄스안에서 다중프로쎄스나 다중스레드를 사용하는 응용프로그람을 구축할수 있다. 결과 다중처리기식조작체계는 다중프로그람처리체계의 모든 기능과 함께 다중처리기를 수용하는 추가적인 기능들을 보장해야 한다. 주요설계문제들중에는 다음과 같은것들이 있다. 즉

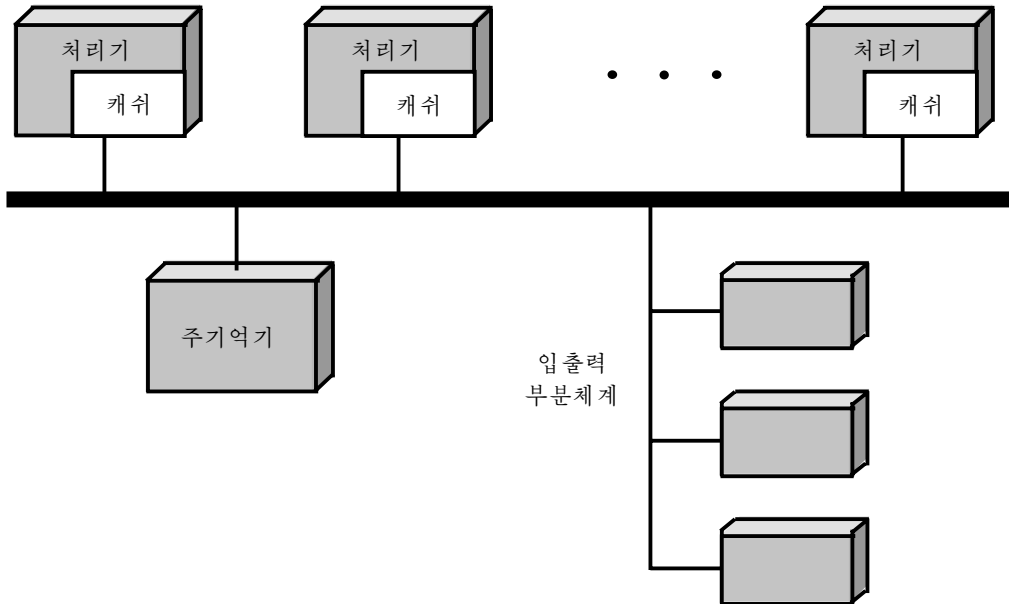



그림 4-9. 대칭형다중처리기구성

- **동시적인 병행프로쎄스 또는 스레드 :** 핵심부루틴은 여러 처리기들이 같은 핵심부코드를 동시에 집행하도록 하기 위해 재진입하여야 한다. 같거나 다른 핵심부의 부분을 집행하는 여러개의 처리기들에 대해 핵심부표와 관리구조를 합리적으로 관리하여 교착이나 무효조작을 피해야 한다.
- **일정작성 :** 일정작성은 임의의 처리기가 수행할수 있으며 그런데로부터 충돌을 피해야 한다. 만일 핵심부준위의 다중스레드처리를 사용하면 그때 다중처리기상에서 동시에 같은 프로쎄스로부터 다중스레드를 일정작성하는데 좋은 기회가 생긴다. 다중처리기의 일정작성은 제10장에서 론의한다.
- **동기화 :** 여러개의 능동프로쎄스들이 공유된 주소공간이나 공유된 입출력자원들에 잠재적으로 접근하는 효과적인 동기화를 보장하는데 주의를 돌려야 한다. 동기화는 호상배제와 사건순서달기를 하게 하는 기능이다. 다중처리기식조작체계에서 쓰이는 공통적인 동기화기구는 제5장에서 서술하는 페쇄이다.
- **기억기관리 :** 다중처리기상에서의 기억기관리는 단일처리기기계에서 나타나는 모든 문제들을 취급해야 하는데 제3편에서 설명한다. 더우기 조작체계는 다중포구기억기와 같은 사용할수 있는 하드웨어병렬기구를 개발하여 가장 좋은 성능을 달성해야 한다. 서로 다른 처리기상에서의 페지화기구를 여러개의 처리기가 페지나 토막을 공유할 때 일관성을 보장하도록 그리고 페지교체에 대해 결심하도록 조정해야 한다.
- **믿음성과 장애극복력 :** 조작체계는 처리기의 고장에 직면하면 일정한 감퇴를 준다. 조작체계의 일정작성기의 다른 부분들은 처리기의 상실을 인식하고 그에 따라 관리들

을 재구조화해야 한다.

다중처리기식조작체계설계문제가 일반적으로 다중프로그람처리의 단일처리기설계 문제에 대한 해결방도에로 확장되므로 다중처리기조작체계를 따로 취급하지 말아야 한다. 오히려 특별한 다중처리기문제를 이 책전반에 걸쳐 적당한 상황에서 설명한다.

제 3 절. 마이크로핵심부

최근에 많은 주의를 돌리고 있는 개념은 마이크로핵심부에 대한 개념이다. 마이크로핵심부는 모듈식확장을 위한 토대를 주는 작은 조작체계의 핵심이다. 그러나 이 말은 어느정도 모호하며 각이한 조작체계설계팀들이 각이하게 대답하는 마이크로핵심부에 대한 몇가지 문제점이 있다. 이 문제점은 얼마나 작은 핵심부에 마이크로핵심부로서의 자격을 주어야 하는가, 하드웨어로부터 그의 기능들을 추상화하는 한편 가장 좋은 성능을 얻기 위하여 장치구동기를 어떻게 설계하는가, 비핵심부조작을 핵심부에서 실행하겠는지 아니면 사용자공간에서 실행하겠는지 그리고 현존부분체계코드(실례로 UNIX의 판본)를 보존하겠는지 아니면 작업코드로부터 출발하겠는지 등과 같은것들을 포함한다.

마이크로핵심부방법은 Mach조작체계에서 그것을 사용하면서 보급되였다. 리론적으로 이 방법은 높은 유연성 및 모듈성을 준다. 마이크로핵심부방법에 대하여 널리 공포하고 사용한 또다른 체계가 W2K인데 이것은 모듈성뿐아니라 이식성도 주요한 리익으로 주장하고 있다. 마이크로핵심부는 몇개의 간결한 부분체계들로 둘러 싸여 있어 여러가지 가동환경상에서 W2K를 실현하는 과업이 쉬워 진다. 몇개의 다른 제품들이 현재 마이크로핵심부를 실현하고 있으며 이 일반적설계방법은 대부분의 개인용콤퓨터, 워크스테이션 및 가까운 앞날에 개발될 봉사기조작체계에서 볼수 있다.

마이크로핵심부구성방식

1950년대 중엽부터 말까지 개발된 초기조작체계들은 구조에 적게 관심을 두고 설계되였다. 실제로 큰 규모의 쏘프트웨어체계를 개발하는데서 누구도 경험이 없었고 호상의존성과 대화에 의하여 일어 난 문제들이 전체적으로 과소평가되였다. **단일화조작체계**에서 실제로 임의의 수속이 임의의 다른 수속을 호출할수 있다. 그러한 구조의 부족은 조작체계가 대량적인 비률로 확장할수 없게 하였다. 실례로 OS/360의 첫 판본은 5000명의 프로그람작성자들이 5년이상에 걸쳐 창조하였고 백만행이상의 코드를 가지고 있었다. 후에 개발된 Multics는 2억행의 코드로 증가하였다. 제2절, 제3절에서 설명한바와 같이 쏘프트웨어개발규모를 조종하는데 모듈식프로그람작성수법이 요구되였다. 특히 **계층조작체계**[7](그림 4-10 ㄱ)가 개발되였으며 여기에서 기능들은 계층구조적으로 구성되여 있으며 대화는 린접한 층들사이에서만 일어 난다. 핵심부방식에서는 대부분 또는 전체적인 층들이 계층식방법으로 집행된다.

계층식방법에도 문제가 있다. 매개 층은 상당히 많은 기능들을 처리한다. 한개 층안에서 대부분의 변화는 린접층(우 및 아래)안에 있는 코드에 많은 영향을 주었고 추적하기가 상당히 힘들었다. 결국 일부 기능들을 더하거나 덜어 낸 기본조작체계의 련속판본

[7] 일반적으로 이 령역에서의 전문용어가 문헌들에서 일관성 있게 적용되지 못하고 있다. 용어 단일화 조작체계는 흔히 단일 및 계층식이라고 하는 두가지 형의 조작체계에 대해 말할 때 쓴다.

들은 실현하기 힘들다. 그리고 린접층들사이에 많은 대화가 있으므로 보안을 실현하기 힘들다.

마이크로핵심부의 기초적인 원리는 절대적으로 본질적인 핵심조작체계기능만이 핵심부안에 있게 된다는것이다. 덜 본질적인 봉사와 응용프로그람들은 마이크로핵심부상에서 실현되며 사용자방식에서 집행한다. 마이크로핵심부안에 있는것과 밖에 있는것들사이에서의 나눔선은 설계에 따라 변하며 공통적인 특징은 전통적으로 조작체계의 일부분으로 되여 온 많은 봉사가 이제는 핵심부와 대화하며 서로 대화하는 외부부분체계라는것이며 이것들은 장치구동기, 파일체계, 가상기억기관리자, 창문체계 및 보안봉사를 가지고 있다.

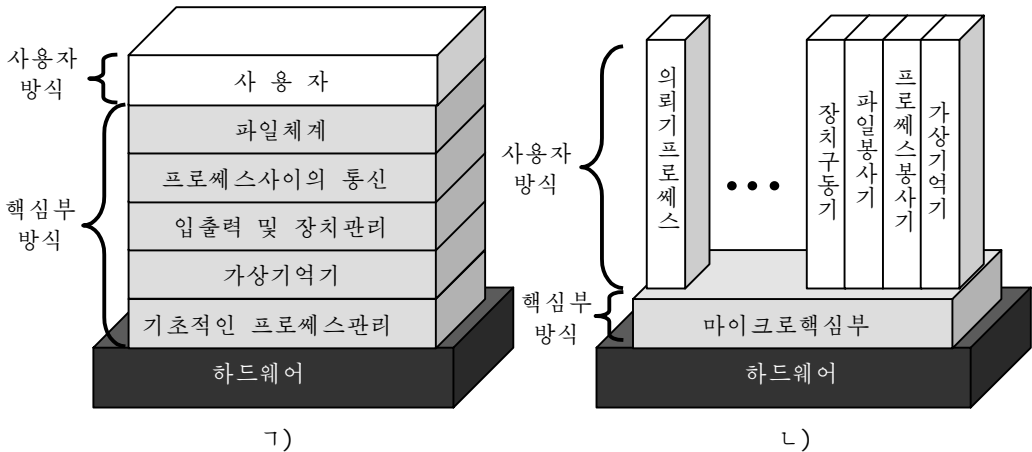



그림 4-10. 핵심부구성방식
ㄱ-계층식핵심부, ㄴ-마이크로핵심부

마이크로핵심부구성방식은 조작체계의 전통적인 수직계층구성을 수평계층구성으로 교체한다(그림 4-10 ㄴ). 마이크로핵심부에 대해 외적인 조작체계구성요소들은 봉사기프로쎄스들로 실현된다. 이 프로쎄스들은 대체로 마이크로핵심부를 통과한 통보문들에 의해 동등한 기초우에서 서로 대화한다. 그러므로 마이크로핵심부는 통보문교환기로서의 기능을 수행한다. 그것은 통보문들을 확인하고 그것들을 구성요소들사이에서 넘겨 주며 하드웨어에 대한 접근을 허락한다. 마이크로핵심부는 또한 보호기능을 수행한다. 즉 교환이 허용되지 않는한 통보문넘기기를 막는다.

실례로 응용프로그람이 어떤 파일을 열자면 통보문을 파일체계봉사기에 보낸다. 만일 그것이 프로쎄스나 스레드를 창조하려고 한다면 프로쎄스봉사기에 통보문을 보낸다. 매개 봉사기들이 통보문들을 다른 봉사기에 보낼수 있으며 마이크로핵심부에서 기본지령기능을 기동시킬수 있다. 이것은 단일콤퓨터안에서의 의뢰기/봉사기구성방식이다.

마이크로핵심부구성의 편리성

마이크로핵심부를 사용하는데서의 몇가지 편리성이 문헌들에 보고되였다(실례로 [IFINK97], [LIED96a], [WAYN94a]). 이것은 다음의것들을 포함한다. 즉

- 통일적인 대면부들
- 확장성

- 유연성
- 이식성
- 믿음성
- 분산체계지원
- 객체지향조작체계(OOOS)에서의 지원

마이크로핵심부설계는 프로쎄스가 제기하는 요청들에 대한 **통일적인 대면부**의 의무를 지니게 한다. 프로쎄스들은 핵심부준위봉사와 사용자준위봉사사이를 식별할것을 요구하지 않는다.

그것은 그러한 모든 봉사가 통보문넘기기에 의해 이루어 지기때문이다. 임의의 조작체계는 새로운 하드웨어장치와 새로운 쏘프트웨어수법이 개발됨에 따라 불가피하게 현재설계에 없는 특징들을 얻을것을 요구한다. 마이크로핵심부구성방식은 **확장성**을 헐하게 하여 같은 기능적범위안에서 추가적인 새로운 봉사는 물론 여러개의 봉사를 준비하게 한다. 실례로 유연성자기원판에서의 다중파일구성이 있을수 있는데 매개 구성은 핵심부에서 가능한 다중파일봉사기들이 아니라 사용자준위프로쎄스로 완성할수 있다. 이렇게 함으로써 사용자는 여러가지 봉사들중에서 사용자의 요구에 가장 적합한 봉사를 선택할수 있다. 마이크로핵심부구성방식에 새로운 기능을 추가할 때 선택된 봉사기들만 수정하거나 추가해야 한다. 새로운 즉 변경된 봉사기의 영향은 그 체계의 부분모임에 국한된다. 더우기 변경은 새로운 핵심부를 만들것을 요구하지 않는다.

마이크로핵심부구성방식의 확장성과 관련되여 있는것은 그의 **유연성**이다. 새로운 기능들을 조작체계에 추가할수 있을뿐아니라 현존기능들을 덜어 내여 더 작고 보다 효과적인 조작체계를 완성할수 있다. 마이크로핵심부에 기초한 조작체계가 필수적으로 작은 체계는 아니다. 실지로 그 구조는 그자체가 넓은 범위의 기능들을 추가할수 있게 해준다. 그러나 누구나 다 실례를 들면 높은 준위의 보안이나 분산식계산작업능력을 요구하는것은 아니다. 만일 본질적인(기억기관리의 견지에서) 기능들을 보조적으로 선택할수 있게 한다면 기본제품은 보다 광범한 각이한 사용자들의 흥미를 끌수 있다.

콤퓨터가동환경시장의 많은 토막들에 대한 Intel회사의 독점이 무한히 계속되지는 않는다. 이런데로부터 **이식성**이 조작체계의 매력 있는 특징으로 되고 있다. 마이크로핵심부구성방식에서 모든 또는 거의 모든 처리기의 특정한 코드는 마이크로핵심부안에 있다. 그래서 체계를 새로운 처리기에 이식하는데서 요구되는 변화는 거의 없고 론리적인 그룹을 이루는 형식으로 배렬되는 경향이 있다.

쏘프트웨어의 규모가 커질수록 그의 믿음성을 담보하는 문제는 점점 더 어려워 진다. 모듈식설계가 **믿음성**을 제고하는데 도움을 주지만 마이크로핵심부구성방식으로 보다 더 큰 믿음성을 달성할수 있다. 작은 마이크로핵심부를 엄밀하게 시험할수 있다. 작은 수의 응용프로그람처리대면부(API)에서 그것을 사용하면 핵심부밖에서 조작체계봉사를 위한 질좋은 코드를 생성할수 있다. 체계프로그람작성자는 주처리기에 대해 제한된 수의 API와 주처리기와의 제한된 대화수단을 가지며 따라서 다른 체계구성요소들에 상반되게 영향을 미친다.

마이크로핵심부는 그자체에 분산형조작체계가 조종하는 클라스터들을 포함하는 **분산체계지원**을 준다. 의뢰기로부터 봉사기프로쎄스에로 통보를 보낼 때 통보는 요청되는 봉사에 대한 식별자를 가지고 있어야 한다. 만일 모든 프로쎄스와 봉사들이 단일한 식별자를 가지도록 분산체계(실례로 클라스터)가 구성된다면 사실상 마이크로핵심부준위에는 단일체계의 상이 있다. 프로쎄스는 어느 기계대상의 봉사가 상주하고 있는지 모르고 통보문을 보낼수 있다. 제6편에서 분산체계를 설명할 때 이 점을 다시 고찰한다.

마이크로핵심부구성방식은 **객체지향조작체계**의 문맥에서 잘 동작한다. 객체지향법은 마이크로핵심부의 설계에 그리고 조작체계에 대한 모듈식확장에 규률을 보태줄수 있다. 그러므로 많은 마이크로핵심부설계노력이 객체지향화방향으로 움직이고 있다[WAYN94b]. 마이크로핵심부구성방식과 OOOS원리를 결합시키는 한가지 유망한 방법은 구성요소들을 사용하는데 있다[MESS96]. 구성요소들은 호상 련결되여 구성요소방식으로 쏘프트웨어를 형성할수 있는 명백히 정의된 대면부를 가진 객체들이다. 구성요소들사이의 모든 대화는 구성요소의 대면부를 사용한다. Windows 2000과 같은 다른 체계들은 전문적으로 즉 완전히 객체지향법에 의거하고 있는것은 아니지만 마이크로핵심부설계에 객체지향원리들을 병합시켰다.

마이크로핵심부의 성능

흔히 인용되는 마이크로핵심부의 잠재적인 한가지 결함은 그의 성능상 결함이다. 마이크로핵심부를 거쳐 통보문을 만들어 보내고 응답을 접수하고 해득하는것은 단일한 봉사호출을 할 때보다 더 오래 걸린다. 그러나 다른 요인들은 성능결함이 있다고 해도 개괄적으로 말하기 힘들게 작용한다.

많은 부분이 마이크로핵심부의 크기와 기능에 관계된다. [LIED 96a]는 첫 세대의 마이크로핵심부라고 부를수 있는것에서의 본질적인 성능상 결함을 로출시키는 몇가지 연구를 요약하고 있다. 이 결함들은 마이크로핵심부코드를 최적화하려는 노력에도 불구하고 지속되여왔다. 이 문제에 대한 하나의 대책은 림계적인 봉사와 구동기들을 조작체계에 다시 집적화하여 넣음으로써 마이크로핵심부를 확대하는것이였다. 이 방법의 초기실례로서는 Mach와 Chorus를 들수 있다. 마이크로핵심부의 기능을 선택적으로 증가시키면 사용자핵심부방식절환수와 주소공간프로쎄스절환수를 감소시킨다. 그러나 이러한 작업과정은 마이크로핵심부설계 즉 최소의 대면부, 유연성 기타 등등을 강화하는 대가로 성능의 결함을 감소시킨다.

다른 방법은 마이크로핵심부를 더 크게가 아니라 더 작게 만드는것이다. [LIED 96b]는 합리적으로 설계한다면 매우 작은 마이크로핵심부가 성능결함을 없애고 유연성과 믿음성을 개선한다고 주장하고 있다. 대표적인 첫 세대 마이크로핵심부는 300kbyte의 코드와 140여개의 체계호출대면부로 구성되여 있다. 작은 2세대 마이크로핵심부의 실례로는 L4[HART97, LIED95]를 들수 있는데 이것은 12kbyte의 코드와 7개의 체계호출로 구성되여 있다. 이 체계들에서의 경험은 그것들이 UNIX와 같은 계층식조작체계만큼 또는 그보다 더 잘 동작할수 있다는것을 보여 주고 있다.

마이크로핵심부설계

각이한 마이크로핵심부들이 일정한 범위의 기능과 크기를 가지고 있으므로 마이크로핵심부가 주는 기능 및 실현되는 구조와 관련하여 그 어떤 고정격식화된 규칙들을 설명할수는 없다. 이 절에서는 최소의 마이크로핵심부의 기능과 봉사의 모임을 보여 주고 마이크로핵심부설계에서의 예비지식을 주게 된다.

마이크로핵심부는 직접적으로 하드웨어에 의존하는 기능들과 봉사기와 사용자방식에서 조작하는 응용프로그람들을 지원하는데 필요한 기능들을 가져야 한다. 이 기능들은 낮은 준위의 기억기관리, 프로쎄스사이의 통신(IPC) 및 입출력새치기관리에 대한 일반적 범주들에 귀착된다.

저준위기억기관리

마이크로핵심부는 프로쎄스준위에서 보호를 실현하기 위해 주소공간에 대한 하드웨어개념을 조종해야 한다. 마이크로핵심부가 매개의 가상페지를 물리적페지프레임에 사영하는데서 책임을 지고 있고 다른 프로쎄스로부터 한 프로쎄스로 주소공간보호, 페지교체알고리듬 및 기타 페지화론리를 포함하는 기억기관리의 대부분은 핵심부밖에서 완성할수 있다. 실례로 마이크로핵심부밖의 가상기억기모듈은 언제 어떤 페지를 주기억기에 가져오며 이미 기억기에 있는 페지를 교체하겠는가를 결심한다. 다음 마이크로핵심부는 페지참조 등을 주기억기의 물리적주소로 변환한다.

페지화 및 가상기억기관리를 핵심부의 외부에서 수행할수 있다는 개념은 Mach의 일부 페지호출프로그람[YOUN87]과 더불어 도입되였다. 그림 4-11은 외부페지호출프로그람의 조작을 설명하고 있다. 응용프로그람에서 스레드가 주기억기안에 없는 페지를 참조하면 페지부재가 발생하며 집행은 핵심부에 대해 내부새치기를 한다. 핵심부는 그다음 어떤 통보문을 페지호출프로그람프로쎄스에 보내여 어느 페지가 참조되였는가를 알려 준다. 페지호출프로그람은 그 페지를 적재하고 그 목적을 위한 어떤 페지프레임을 할당하도록 결정할수 있다. 페지호출프로그람과 핵심부는 페지호출프로그람의 론리적조작을 물리적기억기상에서의 조작으로 변환하기 위해 대화하여야 한다. 일단 그 페지라면 페지호출프로그람은 계속하라는 통보문을 응용프로그람에 보낸다.

이 수법은 비핵심부프로쎄스가 파일들과 자료기지들을 핵심부기동이 없이 사용자주소공간에로 변환할수 있게 한다.

[LIED 95]는 외부페지화 및 가상기억기관리를 지원할수 있는 세가지 모임의 마이크로핵심부조작을 제의하고 있다.

- **허가** : 주소공간에 대한 소유자(프로쎄스)가 몇개의 자기 페지들을 다른 프로쎄스에 넘겨 줄수 있다. 핵심부는 넘겨 주는 프로쎄스의 주소공간에서 페지들을 제거하고 지명된 프로쎄스에 그것들을 할당한다.
- **사영** : 프로쎄스가 임의의 자기페지를 다른 프로쎄스의 주소공간에 사영하여 두 프로쎄스가 모두 그 페지에 접근하도록 할수 있다. 이것은 두 프로쎄스사이에서 공유기억기를 창조한다. 핵심부는 본래 소유자에 페지들을 할당하는것으로 기억하고 있지만 다른 프로쎄스가 접근하는것을 허락해 주는 하나의 방도를 제공하는 것으로 된다.

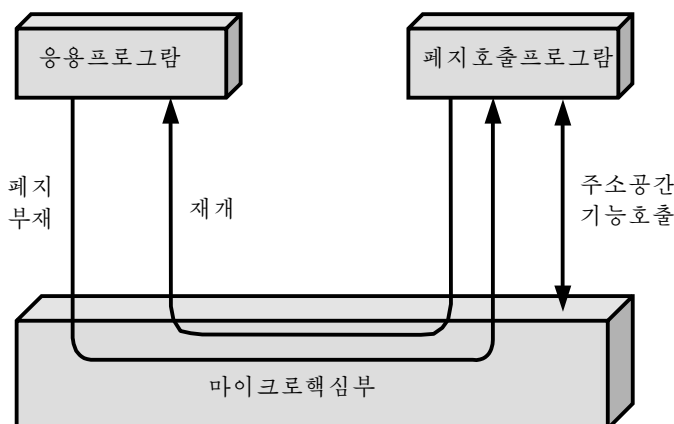



그림 4-11. 페지부재처리

- **되살리기 :** 프로쩨스가 다른 프로쩨스들에 넘겨 주었거나 사영하였던 임의의 페지들을 다시 요구할수 있다.

처음에 핵심부는 모든 물리적기억기를 기본체계프로쩨스가 조종하는 단일주소공간으로 정의한다. 새로운 프로쩨스가 창조됨에 따라 페지들을 원래 총적주소공간으로부터 새로운 프로쩨스에 넘겨 주거나 사영할수 있다. 그러한 방안은 다중가상기억기방안들을 동시에 지원할수 있을것이다.

프로쩨스간 통신

마이크로핵심부조작체계에서 프로쩨스나 스레드사이 통신의 기본형태는 통보문이다. 통보문은 보내고 받는 프로쩨스를 식별해 주는 머리부와 직접적인 자료 또는 그 자료블로크에 대한 지시기나 프로쩨스에 대한 일정한 조종정보를 포함하는 본체를 가지고 있다. 대체로 IPC가 프로쩨스들과 관련된 포구들에 기초하고 있는것으로 생각할수 있다. 포구는 본질상 특정한 프로쩨스에 목적을 둔 통보문들의 대기렬이다. 그 포구와 관련된 내용은 어느 프로쩨스가 이 프로쩨스와 통신할수 있는가를 가리키는 권한에 대한 목록이다.

여기서는 통보문넘기기에 대한 표기가 적당하다. 주소공간을 겹치지 않으면서 개별적프로쩨스들사이에서의 통보문넘기기는 기억기대 기억기복사를 포함하므로 기억기의 속도에 따라 경계가 생기며 처리기속도들을 기준화하지 못한다. 이런데로부터 현재조작체계연구는 스레드에 기초한 IPC와 페지재사영(여러 처리기들이 고유한 단일페지)과 같은 기억기공유방안에서의 흥미 있는 문제들을 반영하고 있다.

입출력 및 새치기관리

마이크로핵심부구성방식으로 하드웨어새치기를 통보문들과 같이 조종할수 있으며 주소공간에서 입출력포구들을 가질수 있다. 마이크로핵심부는 새치기를 인식할수 있으나 그것들을 조종하지는 못한다. 대신 현재 그 새치기와 관련된 사용자준위프로쩨스에 통보문을 내보낸다. 이렇게 함으로써 새치기가 허용되면 특정한 사용자준위프로쩨스를 새치기에 할당하며 핵심부는 사영을 보존한다. 새치기를 통보문으로 변환하는것은 마이크로핵심부가 해야 하지만 마이크로핵심부가 장치특유의 새치기처리에 포함되지는 않는다.

[LIED 96a]는 하드웨어가 통일적인 스레드식별자를 가지며 통보문들(간단히 스레드 ID로 이루어 지는)을 사용자공간의 관련된 쏘프트웨어스레드들에 보내주는 스레드의 모임으로 볼것을 제의하고 있다. 통보문을 수신하는 스레드는 통보문이 새치기로부터 오는것인지 아닌지를 판정하며 특정한 새치기를 판정한다. 그러한 사용자준위코드의 일반적구조는 다음과 같다. 즉

```
 driver thread:
do
waitFor (msg, sender);
if (sender ==  my_hardware_interrupt)
{
                      read/write     I/O ports;
              reset hardware interrupt
}
else ・・・...
while (true);
```

제 4 절. WINDOWS 2000의 스레드 및 SMP관리

Windows 2000(W2K)프로쎄스설계는 여러가지 조작체계환경을 지원할데 대한 요구로부터 만들어 졌다. 여러가지 조작체계가 지원한 프로쎄스들은 다음과 같은것들을 포함하여 방법상 몇가지가 다르다. 즉

- 프로쎄스들의 이름을 붙이는 방법
- 스레드가 프로쎄스에서 주어 지는지 아닌지
- 프로쎄스를 표현하는 방법
- 프로쎄스자원을 보관하는 방법
- 프로쎄스사이 통신과 동기사이에서 사용하는 기구들
- 프로쎄스를 서로 련관시키는 방법

따라서 W2K핵심부가 제공한 본래의 프로쎄스구조와 봉사는 상대적으로 간단하고 일반목적이며 매 조작체계의 부분체계들이 특정한 프로쎄스구조와 기능성을 모의할수 있게 한다. W2K프로쎄스들의 중요한 기능은 다음과 같은것들이다.

- W2K프로쎄스들은 객체들로써 실현된다.
- 집행할수 있는 프로쎄스는 하나이상의 스레드를 가질수 있다.
- 프로쎄스와 스레드객체는 둘다 내장된 동기화가능성을 가진다.

그림 4-12는 프로쎄스를 조종하거나 사용하는 자원과 관련시키는 방법을 설명하고 있다. 매개 프로쎄스는 프로쎄스의 1차통표라고 부르는 보안접근통표를 할당 받는다. 사용자가 처음으로 가입할 때 W2K는 사용자를 위한 보안 ID를 가지는 접근통표를 창조한다. 창조되거나 사용자를 대표하여 실행하는 매개 프로쎄스는 접근통표를 복사하여 가진다. W2K는 통표를 사용하여 사용자가 보안된 체계에 접근하거나 체계상에서 그리고 보안된 객체들상에서 제한된 기능들을 수행할수 있게 해준다. 접근통표는 프로쎄스가 자기자체의 속성을 변화시킬수 있는가 없는가 하는것을 조종한다. 이 경우에 프로쎄스는 자기의 접근통표에 대해 열린 조종을 꼭 가지는것은 아니다. 만일 프로쎄스가 그러한 조종을 열려고 한다면 보안체계는 이것이 허용되는지 그리고 프로쎄스가 자기자체의 속성을 변화시킬수 있는지 없는지 하는것을 판정한다.

프로쎄스와 관련되는것은 또한 프로쎄스에 현재 할당된 가상주소공간을 정의해 주는 블로크의 계렬이다. 프로쎄스는 이 구조를 직접 변경할수 없고 프로쎄스에 기억기배정봉사를 주는 가상기억기관리자에 의거해야 한다.

끝으로 프로쎄스는 객체표를 가지며 그와 함께 프로쎄스에 알려 져 있는 다른 객체들에 대한 조종기를 가진다. 객체에 포함되여 있는 매개 스레드에는 하나의 조종기가 있다. 그림 4-12 는 단일스레드를 보여 준다. 추가적으로 프로쎄스는 파일객체에 대한 그리고 공유된 기억기의 구간을 정의해 주는 구간객체에 접근한다.

프로쎄스와 스레드객체

W2K의 객체지향구조는 일반목적프로쎄스의 기능개발을 촉진시킨다. W2K는 두가지 형태의 프로쎄스관련객체 즉 프로쎄스와 스레드를 사용할수 있게 한다. 프로쎄스는 사용자일감이나 기억기와 같은 자원을 소유하고 파일들을 여는 응용프로그람에 대응하는 실체이다. 스레드는 순차적으로 집행하며 새치기될수 있고 결과 처리기가 다른 스레드에로 전환할수 있게 하는 작업단위이다.

매개 W2K는 프로쎄스를 어떤 객체로 표현하는데 그의 일반구조를 그림 4-13 ㄱ에 보여 주고 있다. 매개 프로쎄스는 몇개의 속성으로 정의되며 수행할수 있는 몇가지 동작이나 봉사들을 묶음으로 하고 있다. 프로쎄스는 해당한 통보문을 접수한 조건에서 어떤 봉사를 수행한다. 단지 그러한 봉사를 기동시키는 방법은 봉사를 주는 프로쎄스객체에 대해 통보문을 사용한다는것이다. W2K가 새로운 프로쎄스를 창조할 때 그것은 새로운 객체의 대상례를 발생시키기 위한 모형으로서 W2K의 프로쎄스용으로 정의되는 객체부류 즉 객체형을 사용한다. 창조당시에 속성값이 할당된다. 표 4-3은 프로쎄스객체에서 매 객체속성에 대한 간단한 정의를 준다.

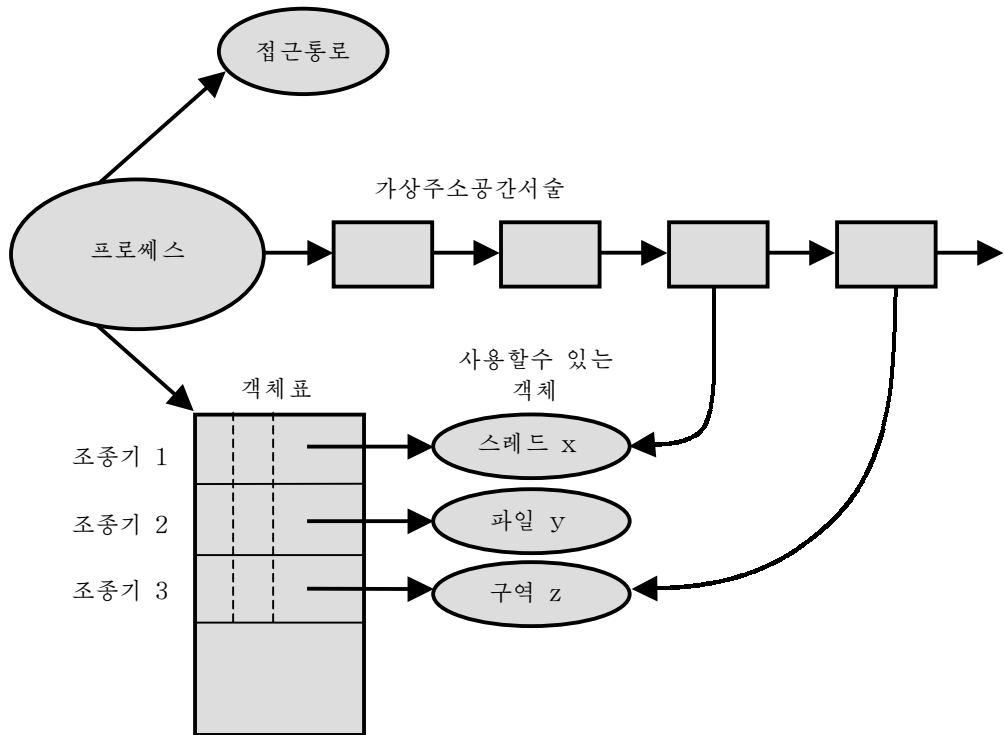



그림 4-12. Windows 2000의 프로쎄스와 그의 자원[CUST 93]

W2K의 프로쎄스는 적어도 하나의 집행할 스레드를 가지고 있어야 한다. 스레드는 다음에 다른 스레드를 창조할수 있다. 다중처리기체계에서 같은 프로쎄스로부터 여러개의 스레드를 병렬로 집행할수 있다. 그림 4-13 ㄴ는 스레드객체에서의 객체구조를 묘사하고 있으며 표 4-4에서는 스레드의 객체구성을 정의한다. 스레드의 일부 속성은 프로쎄스의 속성과 류사하다. 그 경우에 스레드속성값은 프로쎄스의 속성값으로부터 파생된다. 실례로 스레드의 처리기계렬은 처리기의 모임이며 다중처리기체계에서 그것은 스레드를 집행할수 있으며 이 모임은 프로쎄스의 처리기계렬과 등가이거나 프로쎄스처리기계렬의 부분모임이다.

스레드객체의 속성중의 하나가 문맥이라는데 주목하자. 이 정보는 스레드가 중지시키고 집행을 계속할수 있게 한다. 더우기 중지될 때 그의 문맥을 바꾸어 스레드의 성질을 바꿀수 있다.

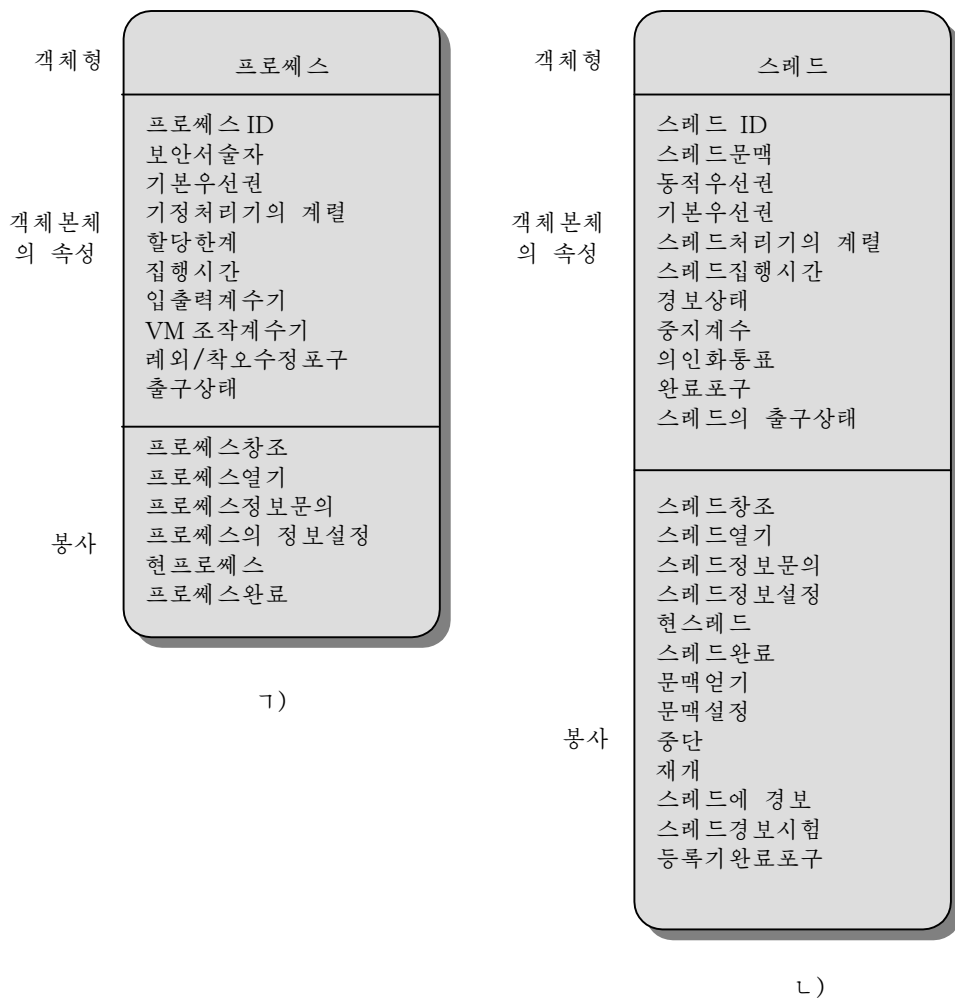



그림 4-13. Windows 2000의 프로쎄스와 스레드의 객체
ㄱ- 프로쎄스객체, ㄴ-스레드객체

표 4-3. Windows 2000프로쎄스의 객체속성

프로쎄스 ID	조작체계에서 프로쎄스를 식별하는 유일한 값
보안서술자	누가 객체를 창조하며 누가 객체에 접근하거나 객체를 사용할 수 있는가 또는 누가 객체에 대한 접근을 무시하는가를 서술한다.
기본우선권	프로쎄스의 스레드들에서의 기본방향집행우선권
기정처리기의 계렬	프로쎄스의 스레드들이 실행할수 있는 기정처리기모임
할당한계	파일공간을 페지화하는 페지화 및 비페지화된 체계
집행시간	기억기의 최대량 및 사용자의 프로쎄스가 사용할수 있는 처리기 시간

입출력계수기	프로쎄스의 모든 스레드가 집행한 총 시간
VM조작계수기	프로쎄스의 스레드가 수행한 입출력조작의 번호와 형을 기록하는 변수
례외/착오수정 포구	프로쎄스의 스레드중 하나가 어떤 례외를 일으킬 때 프로쎄스관리자가 통보문을 보내는 프로쎄스사이의 통신통로들
출구상태	프로쎄스의 완료리유

표 4-4. Windows 2000의 스레드객체속성

스레드 ID	스레드가 봉사기를 호출할 때 스레드를 식별하는 유일한 값
스레드문맥	스레드의 집행상태를 정의하는 등록기값들과 다른 휘발성자료의 모임
동적우선권	임의의 주어 진 순간에서의 스레드집행우선권
기본우선권	스레드의 동적우선권에 대한 보다 낮은 한계
스레드처리기의 계렬	스레드가 실행할수 있는 처리기의 모임으로서 이것은 스레드의 프로쎄스에 대한 처리기계렬의 부분모임 또는 전부
스레드의 집행시간	스레드가 사용자방식 및 핵심부방식에서 집행한 루적시간
경보상태	스레드가 비동기수속호출을 하게 되는가를 가리키는 기발
중단계수	스레드의 집행이 계속 집행되지 않고 중단된 시간
의인화통표	스레드가 다른 프로쎄스대신에(부분체계가 사용함) 조작을 수행하게 하는 림시접근통표
완료포구	프로쎄스관리자가 스레드결속시에(부분체계가 사용함) 통보문을 보내는 프로쎄스사이의 통신통로
스레드의 출구상태	스레드결속리유

다중스레드처리

W2K는 각이한 프로쎄스의 스레드들이 병행으로 집행할수 있기때문에 프로쎄스들사이에서 병행성을 지원한다. 더우기 같은 프로쎄스에서 여러개의 스레드들을 개별처리기들에 배정하여 병행하여 집행할수 있다. 다중스레드식프로쎄스는 여러개의 프로쎄스들을 사용하는데서 간접소비시간이 없이 병행성을 보장한다. 같은 프로쎄스에서 스레드들은 자기들의 공통주소공간을 사용하여 정보를 교환할수 있으며 프로쎄스의 공유자원에 접근한다. 각이한 프로쎄스들안의 스레드들은 두 프로쎄스사이에 설정되여 있는 공유기억기를 통하여 정보를 교환할수 있다.

객체지향다중스레드식프로쎄스는 봉사기의 응용프로그람실현에서 효과적인 수단으로 된다. 실례로 하나의 봉사기프로쎄스가 몇개의 의뢰기를 봉사할수 있다. 매개 의뢰기요청은 봉사기에서 새로운 스레드를 창조한다.

스레드상태

현존 W2K의 스레드는 6개 상태중의 하나에 있다(그림 4-14). 즉

- **준비** : 집행을 위해 일정을 작성할수 있다. 마이크로핵심부의 할당기는 모든 준비상태의 스레드에 대한 추적을 보존하며 우선권순서에 따라 일정을 작성한다.

• **실행대기 :** 실행대기상태의 스레드는 특정한 처리기상에서 다음에 실행하기 위하여 선택되여 있다. 스레드는 이 상태에서 처리기를 받을수 있을 때까지 기다린다. 만일 실행대기상태의 스레드우선권이 충분히 높다면 처리기상에서 실행중인 프로쎄스를 실행대기상태의 스레드가 마음대로 선취할수도 있다. 그렇지 않으면 실행대기상태의 스레드는 실행중인 스레드가 폐색되거나 시간소편을 완료할 때까지 기다린다.

• **실행 :** 일단 마이크로핵심부가 어떤 스레드나 프로쎄스의 절환을 수행하면 실행대기상태의 스레드는 실행상태에 들어 가서 집행을 시작하여 선취되거나 자기의 시간소편을 다 써버리거나 폐색되거나 또는 완료할 때까지 집행을 계속한다. 첫 두 경우에 그것은 준비상태에로 돌아 간다.

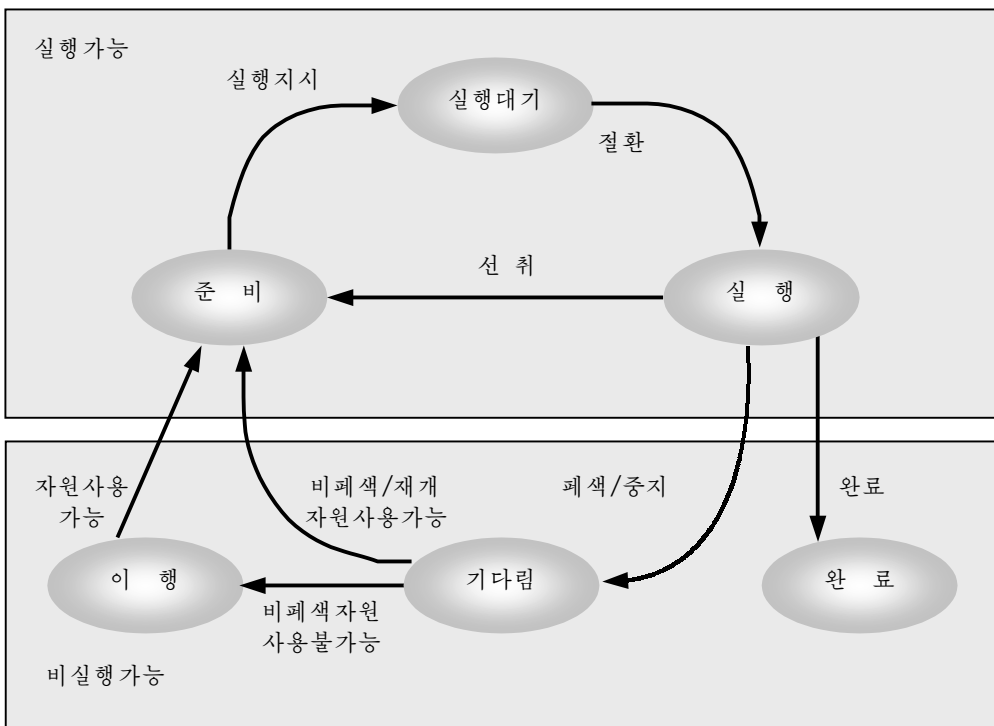



그림 4-14. Windows 2000의 스레드상태

• **기다림 :** 스레드는 (1) 어떤 사건(실례로 입출력사건)에 의해 폐색될 때 (2) 동기화를 위해 자발적으로 기다릴 때 또는 (3) 환경부분체계가 스레드에 그자체를 중단시킬것을 지시할 때 기다림상태에 들어 간다. 기다림조건이 만족될 때 스레드는 그의 모든 자원을 사용할수 있다면 준비상태에로 이동한다.

• **이행 :** 스레드가 실행할 준비는 되였지만 자원을 사용할수 없다면 기다림상태후에 이 상태에 들어 간다. 실례로 스레드의 탄창을 기억기밖으로 페지화할수 있다. 자원을 사용할수 있을 때 스레드는 준비상태로 간다.

• **완료 :** 스레드를 그자체에 의해, 다른 스레드에 의해 또는 그의 부모프로쎄스가 완료될 때 완료시킬수 있다. 일단 보조조작들이 완료되면 스레드는 체계로부터 제거되거나 앞으로 다시 재초기화하기 위해 집행부가 보유할수도 있다.

조작체계의 부분체계들에 대한 지원

일반목적프로쎄스와 스레드의 기능이 여러가지 조작체계의뢰기들의 특정한 프로쎄스 및 스레드의 구조를 지원할수 있다. 조작체계부분체계의 책임은 W2K프로쎄스와 스레드의 기능을 사용하여 그의 대응하는 조작체계에 대한 프로쎄스 및 스레드의 기능들을 모방하는것이다. 프로쎄스/스레드관리령역은 아주 복잡하므로 여기서는 간단한 개괄만 한다.

프로쎄스의 창조는 조작체계응용프로그람으로부터 오는 새로운 프로쎄스에 대한 요청으로부터 시작한다. 새로운 프로쎄스창조요청은 응용프로그람으로부터 대응하는 보호된 부분체계에 보내진다. 다음 부분체계는 프로쎄스의 요청을 W2K집행부에 보낸다. W2K는 프로쎄스객체를 창조하고 부분체계에 대한 객체에 조종을 넘겨 준다. W2K가 프로쎄스를 창조할 때 자동적으로 스레드를 창조하는것은 아니다. Win 32 및 OS/2의 경우에 새로운 프로쎄스는 항상 스레드를 사용하여 창조된다. 따라서 이 조작체계들에서 부분체계는 W2K프로쎄스관리자를 다시 호출하여 새로운 프로쎄스에서의 스레드를 창조한다. 이때 프로쎄스관리자는 W2K로부터 스레드조종을 접수한다. 적당한 스레드 및 프로쎄스정보는 그때 응용프로그람에 반환된다. 16bit Windows 및 POSIX의 경우에 스레드들을 지원하지 않는다. 따라서 이 조작체계에서 부분체계는 프로쎄스를 활성화시키기 위해 W2K로부터 새로운 프로쎄스에서의 스레드를 얻는다. 그러나 프로쎄스정보만은 응용프로그람에 반환한다. 응용프로그람의 프로쎄스가 스레드를 사용하여 실현된다는 사실은 응용프로그람에서 보이지 않는다.

새로운 프로쎄스를 Win 32나 OS/2에서 창조할 때 새로운 프로쎄스는 자기의 많은 속성들을 창조하는 프로쎄스로부터 상속 받는다. 그러나 W2K환경에서 프로쎄스창조는 간접적으로 수행된다. 응용프로그람의 의뢰기프로쎄스는 자기의 프로쎄스창조요청을 조작체계의 부분체계에 내보낸다. 다음 부분체계안의 프로쎄스가 이번에는 프로쎄스요청을 W2K집행부에 내보낸다. 희망하는 효과는 바로 새로운 프로쎄스가 의뢰기프로쎄스의 특성을 계승하고 봉사가 프로쎄스의 속성을 계승하지 않는것이므로 W2K는 부분체계가 새로운 프로쎄스의 부모에게 특성을 부여할수 있게 한다. 새로운 프로쎄스는 그때 부모의 접근통표, 할당한계, 기본우선권 및 기정의 처리기친화성을 계승한다.

대칭형다중처리지원

W2K는 SMP하드웨어구성을 지원한다. 임의의 프로쎄스들의 스레드는 집행부중의 스레드를 포함하여 임의의 처리기상에서 실행할수 있다. 다음단락에서 설명하는 계렬제한이 없는데서 마이크로핵심부는 준비상태의 스레드를 다음번에 사용할수 있는 처리기에 할당한다. 이것은 아무러한 처리기도 휴식할수 없게 하거나 또는 보다 높은 우선권을 가진 스레드가 준비상태에 있을 때 보다 낮은 우선권을 가진 스레드를 집행하지 못하게 해준다. 같은 프로쎄스로부터 여러개의 스레드가 여러 처리기상에서 동시에 집행중에 있을수 있다.

기정사실로서 마이크로핵심부는 스레드를 처리기들에 할당하는데서 유연한 친화적방책을 사용한다. 즉 할당기는 마감에 실행한 같은 처리기에 준비상태의 스레드를 할당하려고 한다. 이것은 스레드의 이전 집행으로부터 처리기의 기억기캐쉬안에 아직 있는 자료를 사용할수 있게 해준다. 응용프로그람이 일정한 처리기들에서 자기 스레드의 집행을 제한할수 있다(딱딱한 친화성).

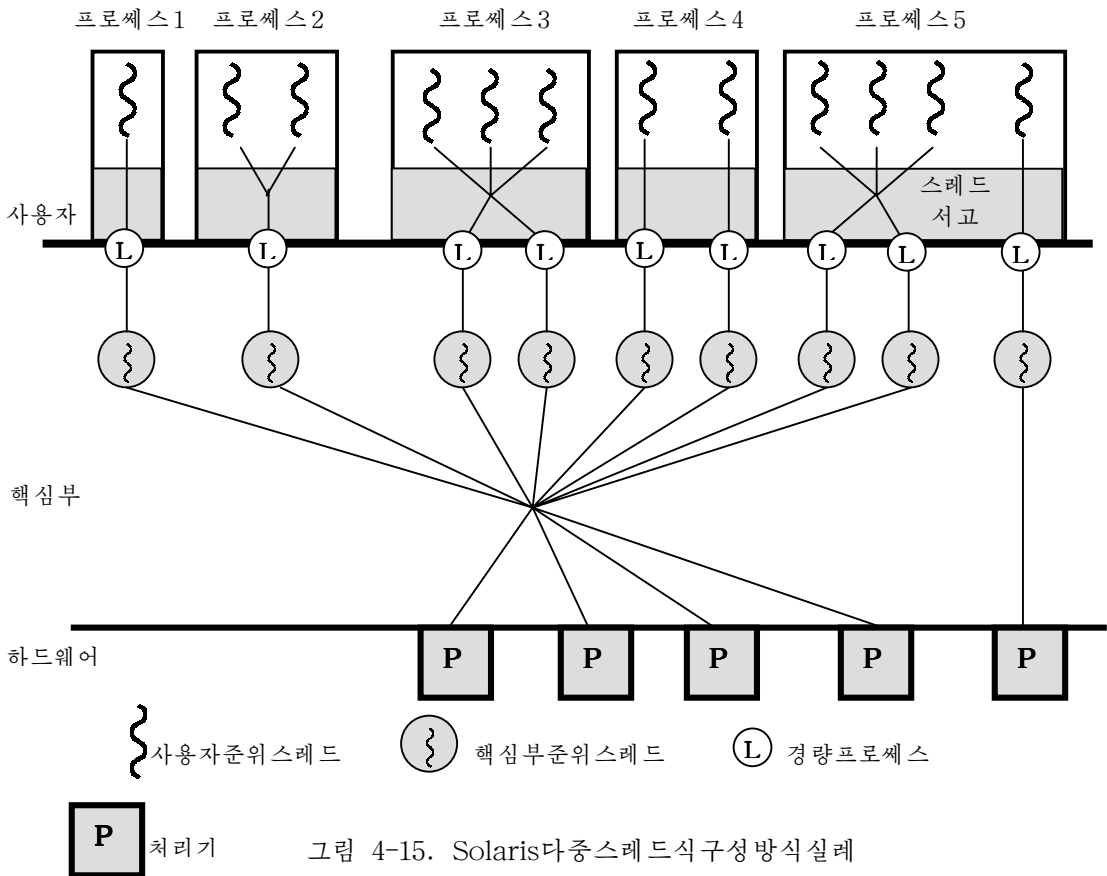
제 5 절. Solaris의 스레드 및 SMP관리

Solaris는 처리기의 자원을 개발하는데서 일정한 유연성을 주기 위해 설계된 례외적인 다중준위스레드지원을 실현한다.

다중스레드식구성방식

Solaris는 4개의 개별적인 스레드관련개념들을 사용한다.

- **프로쎄스 :** 이것은 보통 UNIX프로쎄스이며 사용자의 주소공간탄창 및 프로쎄스조종블로크를 가진다.

그림 4-15. Solaris다중스레드식구성방식실례

- **사용자준위스레드 :** 프로쎄스의 주소공간에 있는 스레드서고를 통하여 실현되므로 이것들은 조작체계에서 보이지 않는다. 사용자준위스레드(ULT)[8]는 응용프로그람의 기구에서의 병렬기구를 위한 대면부이다.
- **경량프로쎄스 :** 경량프로쎄스(LWP)는 ULT와 핵심부스레드사이에서의 사영으로 볼수 있다. 매개 LWP는 하나이상의 ULT를 지원하며 하나의 핵심부스레드에로 사영한다. LWP는 핵심부가 독립적으로 일정을 작성하며 다중처리기상에서

8. 략자 ULT 는 필자가 붙인것이며 Solaris 문헌에서도 찾아 볼수 없다.

병렬로 집행할수 있다.

- **핵심부스레드 :** 이것들은 체계처리기중의 한 처리기상에서 실행하도록 일정을 작성하고 할당할수 있는 기본실체들이다.

그림 4-15는 이 4개의 실체들사이의 관계를 설명하고 있다. 매개 LWP를 위한 하나의 핵심부스레드가 언제나 정확히 있다는것을 주목하자. LWP는 응용프로그람의 프로쎄스에서 볼수 있다. 그래서 LWP자료구조는 자기의 개별적인 프로쎄스주소공간에 존재한다. 동시에 매개 LWP는 단일할당가능한 핵심부스레드에 대해 경계를 짓게 되며 핵심부스레드에서의 자료구조는 핵심부의 주소공간에 보존된다.

실례에서 프로쎄스 1은 단일 LWP에 대해 경계를 지은 단일한 ULT로 구성되여 있다. 이리하여 전통적인 UNIX프로쎄스에 대응하는 단일스레드의 집행이 있다. 단일프로쎄스에서 병행성이 요구되지 않을 때 응용프로그람은 프로쎄스구조를 사용한다. 프로쎄스 2는 순수한 ULT전략에 대응한다. 단일핵심부스레드가 모든 ULT를 지원하며 따라서 한번에 오직 하나의 ULT를 집행할수 있다. 이 구조는 병행성을 표현하는 방법으로 프로그람처리를 제일 잘할수 있는 응용프로그람에서 쓸모가 있으나 여러 스레드들을 병렬적으로 집행할 필요는 없다. 프로쎄스 3은 보다 적은 수의 LWP상에서 다중화된 여러 스레드를 보여 주고 있다. 일반적으로 Solaris는 응용프로그람들이 보다 적거나 같은 수의 LWP상에서 ULT들을 다중화할수 있다. 이것은 응용프로그람이 프로쎄스를 지원하게 될 핵심부준위에서의 병렬화정도를 특징 짓는다. 프로쎄스 4는 1대 1사영에서 LWP에 경계를 지은 자기의 스레드를 영구적으로 가진다. 이 구조는 핵심부준위병렬화가 응용프로그람에 충분히 보일수 있게 해준다. 스레드들이 대체로 또는 흔히 페색중 방식으로 중단된다면 그것이 쓸모가 있다. 프로쎄스 5는 여러 LWP상에서의 여러 ULT사영과 LWP에 대한 ULT의 경계 짓는 수법을 모두 보여 주고 있다. 추가적으로 하나의 LWP는 특정한 처리기에서 경계를 짓는다.

그림에서는 LWP와 관련이 없는 핵심부스레드의 존재를 보여 주지 않고 있다. 핵심부는 핵심부스레드를 창조, 실행 및 파괴함으로써 특정한 체계기능을 집행한다. 체계기능을 실현하기 위해 핵심부프로쎄스가 아니라 핵심부스레드를 사용하면 핵심부에서의 절환(프로쎄스절환으로부터 스레드절환에로의)에 대한 간접소비시간을 감소시킨다.

동기

사용자준위 및 핵심부준위스레드들의 결합은 응용프로그람작성자에게 주어 진 응용프로그람에서 가장 효과적이며 가장 적당한 방법으로 병행성을 개발할수 있게 한다.

일부 프로그람들은 간단화를 위해 개발할수 있는 론리적인 병렬기구와 구조식코드를 가지지만 하드웨어적인 병렬기구를 요구하지는 않는다. 실례로 응용프로그람이 여러 창문들을 사용하는데 어떤 시간에 그중의 하나만이 능동상태로 되며 그 응용프로그람은 단일 LWP상에서 ULT의 모임처럼 실현되는 우점을 가질수 있다. 그러한 응용프로그람들을 ULT들로 국한시키는것이 가지는 우점은 효과성이다. ULT는 핵심부를 기동시키지 않고 창조, 파괴, 페색, 활성화 등을 할수 있다. 매개 ULT가 핵심부에 알려 진다면 핵심부는 핵심부자료구조를 매개 ULT에 배정해야 하며 스레드를 절환해야 할것이다. 알고 있는바와 같이(표 4-1) 핵심부준위의 스레드절환은 사용자준위의 스레드절환보다 비용이 더 든다.

만일 응용프로그람이 입출력을 수행하고 있을 때와 같이 페색될수 있는 스레드를 가지고 있다면 여러개의 LWP들을 가지고 같거나 더 많은 수의 ULT들을 지원하는것이

퍽 유리할것이다. 응용프로그람도 스레드서고요구도 같은 프로쎄스에서 다른 스레드가 집행하도록 외곡하지 않는다. 대신 프로쎄스안에서 한 스레드가 페색된다면 프로쎄스의 다른 스레드들은 남아 있는 LWP들상에서 실행할수도 있다.

ULT들을 LWP들에로 1대 1사영하는것은 일부 응용프로그람에서 효과적이다. 실례로 어떤 병렬배렬계산은 그 배렬의 렬들을 서로 다른 스레드들사이에서 나눌수 있을것이다. LWP당 정확히 하나의 ULT가 있다면 진행되는 계산에서 아무러한 스레드절환도 필요 없다.

LWP들에 영구적으로 경계를 지은 스레드들과 경계를 짓지 않는 스레드들(여러 LWP들을 공유하고 있는 여러 스레드들)을 혼합시키는것은 일부 응용프로그람에서 합리적이다. 실례로 실시간응용프로그람의 일부 스레드들이 체계범위의 우선권과 실시간일정작성을 하며 한편 다른 스레드들이 기본기능을 수행하며 하나 또는 그보다 작은 LWP들의 동적구역을 공유할수 있도록 할것을 원할수도 있다.

프로쎄스구조

그림 4-16은 일반적인 용어들로 전통적인 UNIX체계의 프로쎄스구조와 Solaris 의 프로쎄스구조를 비교한다. 대표적인 UNIX실현물을 보면 프로쎄스구조는 처리기 ID, 사용자 ID, 신호할당표를 포함하는데 핵심부는 이것을 사용하여 프로쎄스에 신호를 보낼 때 무엇을 하겠는가를 결정하며 또한 파일서술자를 포함하는바 그것은 프로쎄스가 사용하는 파일들의 상태를 서술해 준다. 그리고 프로쎄스구조는 기억기배치표를 포함하고 있는데 이것은 프로쎄스에서의 주소공간을 정의해 주며 또한 처리기의 상태구조를 포함하는바 이것은 프로쎄스용핵심부탄창을 포함한다. Solaris는 이 기본구조를 보존하지만 처리기의 상태블로크를 매개 LWP를 위한 하나의 자료블로크를 가지는 구조들의 목록으로 교체한다.

LWP자료구조는 다음과 같은 요소들을 가지고 있다. 즉

- LWP식별자
- LWP의 우선권 및 이로부터 그것들을 지원하는 핵심부스레드
- 신호를 접수하게 될 핵심부를 알려 주는 신호가리개
- 보관된 사용자준위등록기들의 값(LWP가 실행중에 있지 않을 때)
- LWP에서의 핵심부탄창은 체계호출독립변수, 결과 및 매 호출준위에서의 오유코드를 가지고 있다.
- 자원의 사용 및 프로그람실행통보자료
- 대응하는 핵심부스레드에 대한 지시기
- 프로쎄스의 구조에 대한 지시기

스레드집행

그림 4-17은 ULT와 LWP량자의 집행상태를 간소화하여 보여 주고 있다. 사용자준위스레드들의 집행은 스레드서고가 관리한다. 우선 경계를 짓지 않는 스레드 즉 많은 LWP들을 공유하는 스레드를 고찰하자. 경계를 짓지 않은 스레드는 4개의 상태 즉 실행가능, 능동, 재우는 및 정지된 상태들중의 하나에 있을수 있다. 능동상태에서 ULT는 일반적으로 기초로 되는 핵심부스레드가 집행하는 동안 LWP에 할당되여 집행한다. 몇가

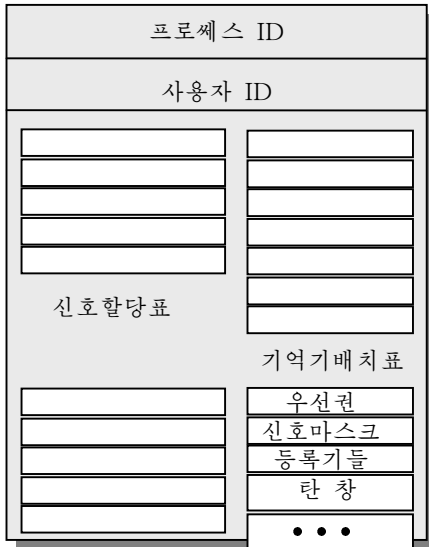
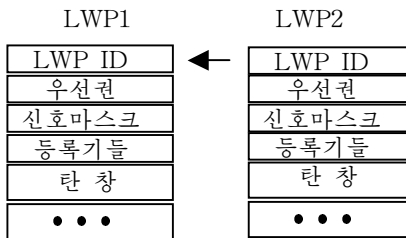
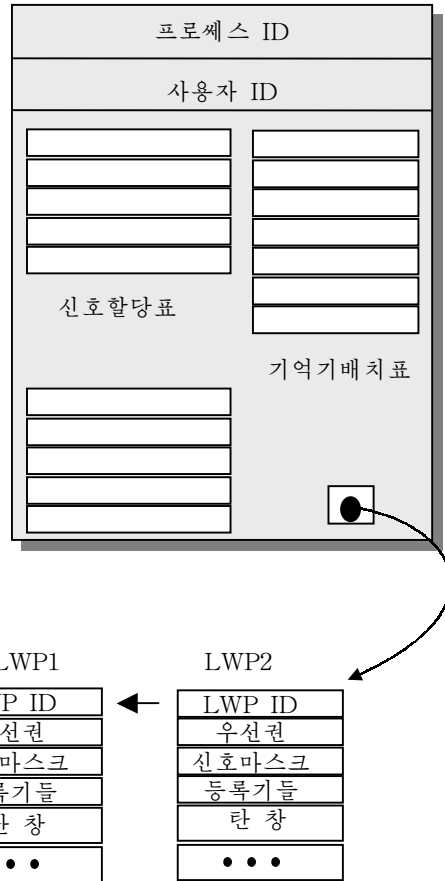



그림 4-16. 전통적인 UNIX와 Solaris 2.x에서 프로쎄스의 구조[LEWI9.6]

지 사건들이 ULT를 능동상태에서 벗어 나게 할수 있다. T1이라고 부르는 능동 ULT를 고찰하자. 다음의 사건들이 발생할수 있다. 즉

- **동기화 :** T1는 제5장에서 설명하는 병행성의 기본지령중의 하나를 기동하여 그의 동작을 다른 스레드들과 함께 조종하도록 하며 호상 배제하도록 한다. T1은 재우는 상태에 놓인다. 동기화조건이 만족되면 T1은 실행가능한 상태로 이동한다.
- **중단 :** T1을 중단시켜 임의의 스레드(T1을 포함하여)를 정지된 상태에 놓이게 할수 있다. T1은 다른 스레드가 실행가능한 상태로 이동시켜 줄 요청을 계속 내보낼 때까지 그 상태에 머물러 있는다.
- **선취 :** 능동스레드(T1 또는 일정한 다른 스레드)는 보다 높은 우선권을 가진 다른 스레드(T2)를 실행가능상태로 되도록 해주는 어떤것을 한다. 만일 T1이 가장 낮은 우선권을 가진 능동스레드라면 그것은 선취되여 실행가능한 상태로 이동하고 T2가 사용할수 있게 된 LWP에 할당된다.
- **양보 :** 만일 T1이 thr-yield()의 서고지령을 집행한다면 서고에서 스레드의 일정작성프로그람은 같은 우선권을 가진 실행가능한 스레드(T2)가 있는가를 본다. 만

일 있다면 T1이 실행가능한 상태에 놓이고 T2가 사용할수 있게 된 LWP에 할당된다. 만일 그렇지 않으면 T1이 계속 실행한다.

선행한 모든 경우들에서 T1이 능동상태밖으로 이동하면 스레드서고는 실행가능한 상태에 있는 또 다른 경계짓지 않은 스레드를 선택하여 그것을 새롭게 사용할수 있는 LWP상에서 실행한다.

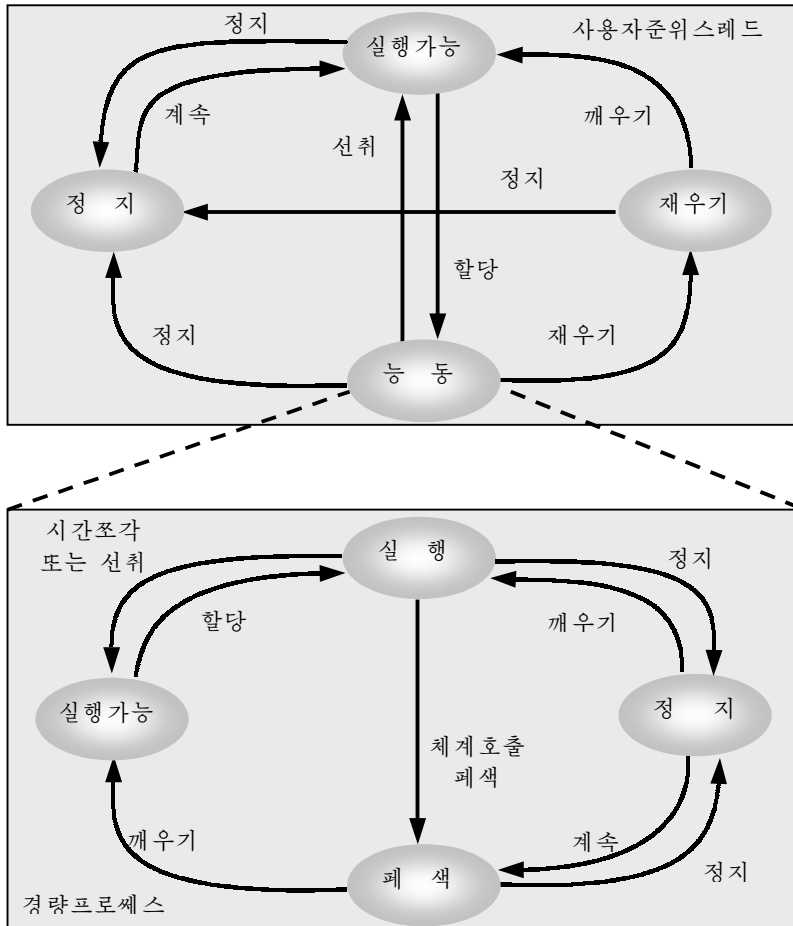



그림 4-17. Solaris 의 사용자준위스레드와 LWP 의 상태

그림 4-17은 모든 LWP에서의 상태도를 보여 주고 있다. 우리는 이 상태도를 ULT 의 능동상태에 대한 세부적인 서술로 볼수 있다. 그것은 경계 짓지 않은 스레드가 오직 능동상태에 있을 때 그에 해당한 LWP를 가지고 있기때문이다. LWP상태도는 합리적으로 자체해석적이다. 능동스레드는 LWP가 실행중 상태에 있을 때에만 집행중에 있다. 능동스레드가 페색중인 체계호출을 집행할 때 LWP는 페색상태에 들어 간다. 그러나 ULT가 LWP에 경계를 지은채로 있고 스레드서고가 관련되여 있을 때까지 ULT는 능동상태를 보존한다.

경계 지은 스레드에서 ULT와 LWP사이의 관계는 약간 다르다. 실례로 경계 지은 ULT가 동기화사건을 기다리고 있는 재우는 상태에로 이동한다면 LWP도 역시 실행을 정지해야 한다. 이것은 핵심부준위의 동기화변수에 기초하여 LWP블로크를 가지는것으로 성취된다.

스레드로서의 새치기

대부분의 조작체계들은 두개의 기본적인 비동기동작형태 즉 프로쎄스와 새치기를 포함한다. 프로쎄스들(또는 스레드)은 서로 협동하며 여러가지 기본지령에 의하여 공유된 자료구조의 사용을 관리한다. 기본지령은 호상배제(한번에 하나의 프로쎄스만이 일정한 코드를 집행하거나 일정한 자료에 접근할수 있다.)를 실시하며 그것들의 집행을 동기화시킨다. 새치기는 어떤 시간주기동안 그것들의 발생을 차단시킴으로써 동기화된다. Solaris는 이 두 개념을 단일방식 즉 핵심부스레드 및 핵심부스레드를 일정작성하고 집행하기 위한 기구들로 통일화한다. 이것을 하기 위해 새치기를 핵심부스레드로 변환한다.

새치기를 스레드로 변환하는데서의 동기는 간접소비시간을 감소시키자는것이다. 새치기조종기는 흔히 핵심부의 나머지부분이 공유한 자료를 조작한다. 따라서 그러한 자료에 접근하는 핵심부루틴을 집행하는 동안 대부분의 새치기가 그 자료에 영향을 미치지 않는다할지라도 새치기는 페색되여야 한다. 대체로 이것을 수행하는 방법은 루틴이 새치기를 페색시키기 위해 보다 높은 새치기우선권준위를 설정하며 접근이 완수되면 보다 낮은 우선권준위를 설정하는것이다. 이 문제는 다중처리기체계에 기초하여 확대된다. 핵심부는 객체들을 더 잘 보호해야 하며 모든 처리기들에서 새치기를 페색시키도록 요구할수도 있다.

Solaris에서 해결대책은 다음과 같이 요약할수 있다. 즉

1. Solaris는 핵심부스레드의 모임을 사용하여 새치기를 조종한다. 임의의 핵심부스레드에 대해서와 마찬가지로 새치기스레드는 자체식별자, 우선권, 문맥 및 탄창을 가진다.
2. 핵심부는 자료구조에 대한 접근을 조종하며 제5장에서 서술되는 형태중에서 호상배제의 기본지령들을 사용하여 새치기스레드들사이에서 동기화를 보장한다. 즉 스레드들에서의 보통 동기화수법을 새치기를 처리하는데 사용한다.
3. 새치기스레드는 모든 다른 형의 핵심부스레드들보다 높은 우선권들을 할당 받는다.

새치기가 발생할 때 그것은 특정한 처리기에 배당되며 처리기상에서 집행되고 있던 스레드는 고정된다. 고정된 스레드는 다른 처리기에 이동할수 없으며 그의 문맥은 보존된다. 그것은 단순히 새치기가 처리될 때까지 새치기된다. 처리기는 그다음 새치기스레드를 집행하기 시작한다. 사용할수 있는 활성화를 상실한 새치기스레드들의 저장구역이 있어 새로운 스레드창조가 필요 없다. 새치기스레드는 다음에 집행하여 그 새치기를 처리한다. 만일 처리기루틴이 다른 집행중인 스레드가 사용하도록 일정한 류형으로 현재 고정되여 있는 자료구조에 대한 접근을 요구한다면 새치기스레드는 접근을 기다려야 한다. 새치기스레드는 단지 보다 높은 우선권을 가진 다른 새치기스레드가 선취할수 있다.

Solaris 새치기스레드에서의 경험은 이 방법이 전통적인 새치기처리전략에서 우월하다는것을 보여 준다.

제 6 절. LINUX의 프로쎄스 및 스레드관리

LINUX의 프로쎄스

프로쎄스나 과제를 LINUX에서는 task_struct 자료구조로 표현한다. LINUX는 task 표를 가지고 있는데 이것은 현재 정의되여 있는 매 task_struct 자료구조에 대한 선형벡토르로 되여 있는 지시기이다. task_struct 자료구조는 몇가지 범주에 속하는 정보를 가지고 있다.

- **상태 :** 프로쎄스의 집행상태(집행중, 준비, 중단, 정지, 좀비)이다. 이것은 후에 설명한다.
- **일정작성정보 :** 프로쎄스의 일정을 작성하기 위해 LINUX가 요구하는 정보이다. 프로쎄스는 보통 또는 실시간프로쎄스로 될수 있으며 일정한 우선권을 가진다. 실시간프로쎄스들은 보통프로쎄스들보다 먼저 일정을 받으며 매 범주에서 상대적인 우선권을 사용할수 있다. 계수기는 프로쎄스를 집행하는데 허용되는 시간의 추적을 보존한다.
- **식별자 :** 매 프로쎄스는 유일한 프로쎄스식별자를 가지며 또한 사용자 및 그룹식별자를 가진다. 그룹식별자는 사용자가 속한 그룹에 자원접근권한을 할당하는데 사용된다.
- **프로쎄스간통신 :** LINUX는 UNIX SVR4에서 보게 되는 IPC기구를 지원하는데 이 문제는 제6장에서 서술한다.
- **련결지령 :** 매개 프로쎄스는 자기의 부모프로쎄스에 대한 련결지령, 자기의 자식들(같은 부모를 가지는 프로쎄스들)에 대한 련결지령들 및 그의 모든 자식들에 대한 련결지령을 가진다.
- **시간과 시계 :** 프로쎄스의 창조시간과 프로쎄스가 지금까지 소비한 처리기의 시간을 포함한다. 프로쎄스는 또한 하나이상의 간격시계들과 관련되여 있을수 있다. 프로쎄스는 체계호출로써 간격시계를 정의하며 결과적으로는 시계가 다 돌아 갔을 때 프로쎄스에 신호를 보낸다. 시계는 한번 사용할수도 있고 주기적으로 사용할수도 있다.
- **파일체계 :** 프로쎄스가 열어 놓은 임의의 파일에 대한 지시기들을 가지고 있다.
- **가상기억기 :** 프로쎄스에 할당된 가상기억기를 정의한다.
- **처리기 특정의 문맥 :** 프로쎄스에 대한 문맥을 구성하는 등록기 및 탄창정보

그림 4-18은 프로쎄스의 집행상태를 보여 주고 있다. 그 상태는 다음과 같다. 즉

- **실행중 :** 이 상태의 값들은 두 상태에 대응한다. 실행중 프로쎄스는 집행중이나 집행할 준비상태중의 어느 한 상태에 있다.
- **새치기가능 :** 이것은 폐색된 상태로서 이 상태에서 프로쎄스는 입출력조작을 끝낸다. 자원의 사용가능성 또는 다른 프로쎄스로부터의 신호와 같은 어떤 사건을 기다리고 있다.
- **새치기불가능 :** 이것은 또 하나의 폐색된 상태이다. 이것과 새치기가능상태사이의 차이는 새치기불가능상태에서 프로쎄스가 하드웨어조건을 직접 기다리고 있으며 따라서 어떤 신호도 받으려고 하지 않는다.
- **정지 :** 프로쎄스는 정지되여 있고 다른 프로쎄스로부터의 실제 동작에 의해서만 계속 집행할수 있다. 실례로 착오수정을 받고 있는 프로쎄스를 정지상태에 놓을수 있다.

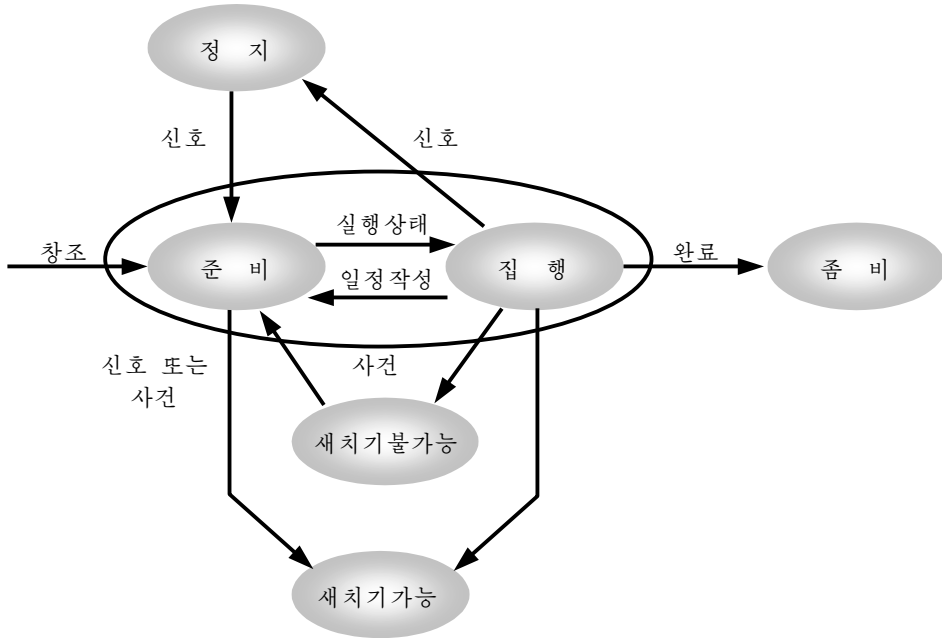



그림 4-18. Linux의 프로쎄스/스레드모형

- **좀비 :** 프로쎄스가 완료하였지만 일정한 리유로 하여 아직 자기의 과제구조를 프로쎄스표에 가지고 있어야 한다.

Linux의 스레드

새로운 프로쎄스가 현재의 프로쎄스속성들을 복사하여 Linux안에서 창조된다. 파일, 신호조종기들 및 가상기억기와 같은 자원들을 공유하도록 새로운 프로쎄스를 복제할수 있다. 두 프로쎄스가 같은 가상기억기를 공유할 때 그것들은 단일프로쎄스의 스레드들과 같이 동작한다. 그러나 그 어떤 개별적형태의 자료구조도 스레드용으로 정의되지 않는다. 그리하여 Linux는 스레드와 프로쎄스사이에 그 어떤 차이도 두지 않는다.

요약, 기본용어 및 복습문제

일부 조작체계들은 프로쎄스와 스레드에 대한 개념을 명백히 구별하는데 전자는 자원소유권과 관련되여 있고 후자는 프로그람의 집행과 관련되여 있다. 이 방법은 효과성 및 코드화의 편리성을 개선할수 있다. 다중스레드체계에서 여러 병행스레드들은 단일프로쎄스에서 정의할수 있다. 사용자준위스레드는 조작체계에 알려 지지 않으며 프로쎄스의 사용자공간에서 실행하는 스레드서고가 창조하고 관리한다. 사용자준위스레드는 방식절환이 한 스레드로부터 다른 스레드로 절환할것을 요구하지 않기때문에 매우 효과적이다. 그러나 프로쎄스의 단일사용자준위스레드만 한번에 집행할수 있고 만일 한 스레드가 페색되면 전체 스레드가 페색된다. 핵심부준위스레드는 핵심부가 보존하고 있는 프로쎄스의 스레드들이다. 핵심부가 그것들을 인식하기때문에 같은 프로쎄스의 여러 스레드를 다중처리기상에서 집행할수 있으며 스레드페색이 전체 프로쎄스를 페색시키지 않는다. 그러나 방식절환은 한 스레드로부터 다른 스레드에로 절환할것을 요구한다.

대칭형다중처리는 임의의 프로쎄스(또는 스레드)가 임의의 처리기상에서 실행할수 있는 그러한 다중처리기체계를 조직하는 방법으로서 이것은 핵심부코드와 프로쎄스를 가진다. SMP구성방식은 새로운 조작체계설계문제를 제기하며 류사한 조건하에서 단일처리기체계보다 더 큰 성능을 준다.

최근년간에 조작체계설계와 관련하여 마이크로핵심부방법에 많은 관심을 두고 있다. 그의 순수한 형식에서 마이크로핵심부조작체계는 핵심부방식에서 실행하며 가장 본질적이며 림계적인 조작체계기능만을 가지는 매우 작은 마이크로핵심부로 구성되여 있다. 다른 조작체계기능들은 사용자방식에서 집행하도록 그리고 림계봉사들에서 마이크로핵심부를 사용하도록 완성된다. 마이크로핵심부설계는 유연하고 높은 모듈식개발을 할수 있게 해준다. 그러나 이러한 구성방식의 성능에 대한 문제점들이 남아 있다.

기본용어

핵심부준위스레드(KLT) 경량프로쎄스 마이크로핵심부 단일화조작체계	다중스레드처리 프로쎄스 대칭형다중처리기 (SMP)	과제 스레드 사용자준위스레드 (ULT)

복습문제

1. 표 3-5는 비스레드식조작체계의 프로쎄스조종블로크에서 보게 되는 대표적인 요소들을 목록화하고 있다. 이중에서 어느것이 스레드조종블로크에 속하며 어느것이 다중스레드식체계에서의 프로쎄스조종블로크에 속하는가?
2. 스레드사이에서의 방식절환이 프로쎄스사이에서의 방식절환보다 비용이 적게 드는 리유를 렬거하시오.
3. 프로쎄스의 개념에 포함된 두개의 개별적이며 잠재적으로 독립적인 특성은 무엇인가?
4. 단일사용자다중프로쎄스처리체계에서 스레드를 사용하는 일반적인 실례를 4개 드시오.
5. 프로쎄스의 모든 스레드들이 대표적으로 공유하게 되는 자원은 무엇인가?
6. KLT들에 대한 ULT들의 세가지 우점들을 렬거하시오.
7. ULT들의 두가지 결함을 KLT와 비교하여 렬거하시오.
8. 쟈케트화를 정의하시오.
9. 그림 4-8에서 이름을 붙인 여러가지 구성방식들을 간단히 정의하시오.
10. SMP조작체계에서의 주요설계문제를 렬거하시오.
11. 마이크로핵심부조작체계에서 외부부분체계들로 될수도 있는 대표적인 단일화조작체계에서의 봉사와 기능들의 실례를 드시오.
12. 마이크로핵심부설계의 7개 우점을 단일화조작체계설계와 비교하여 렬거하고 간단히 설명하시오.
13. 마이크로핵심부조작체계의 성능상 결함을 설명하시오.
14. 극히 작은 마이크로핵심부조작체계에서도 찾아 볼수 있다고 기대하는 세가지 기능을 렬거하시오.
15. 마이크로핵심부조작체계에서 프로쎄스나 스레드들사이에서 통신의 기본형식은 무엇인가?

참 고 문 헌

[LEWI 96] 과 [KLEI 96]은 스레드개념과 프로그람처리전략에 대한 설명을 잘 개괄해 주고 있다. 전자는 개념들에 더 중심을 두고 있고 후자는 프로그람처리에 더 중심을 두고 있으나 량자는 두가지 문제들을 쓸모 있게 포괄하고 있다. [PHAM 96]은 Windows NT스레드기능을 설명하고 있는데 이것은 깊이에 있어서 Windows 2000의 스레드기능들과 동일하다.

[MUKH 96]은 SMP들에서의 조작체계설계문제를 잘 설명하고 있다. [CHAP 97]은 다중처리기식조작체계에서 최근 설계방향에 기초한 다섯개의 기사들을 담고 있다. 마이크로핵심부설계에 대한 가치 있는 설명은 [LIED 95]와 [LIED 96]에 포함되여 있는데 후자는 성능문제에 중심을 두고 있다.

CHAP97 Chapin, S., andMaccabe, A., eds. "Multiprocessor Operating Systems: Harnessing the Power." Special issue of *IEEE Concurrency*, April-June 1997

KLE196 Kleiman, S.; Shah, D.; and Smallders, B. *Programming with Threads*. Upper Saddle River, NJ:Prentice Hall, 1996.

LEW 196 Lewis, B., and Berg, D. *Threads Primer*. Upper Saddle River, NJ:Prentice Hall, 1996.

LIED95 Lidtke, J. "On μ-Kernel Construcion." *Processing of the Fifteenth ACM Symposium on Operating Systems Principles,* December 1995.

LIED96 Liedtke, J. "Toward Real Microkernels." *Communications of the ACM,* September 1996.

MUKH96 Mukherjee, B., and Karstem, S. "Operating Systems for Parallel Machines." *In Parallel Computers: Theory and Practice.* Edited by T. Casavant, P. Tvrkik, and F. Plasil. Los Alamitos, CA:IEEE Computer Society Press, 1996.

PHAM96 Pham, T., and Garg, P. *Multithreaded Programming with Windows NT.* Upper Saddle River, NJ:Prentice Hall, 1996.

련 습 문 제

1. 프로쎄스에서 다중스레드를 사용하는것이 가지는 두가지 우점은 (1) 현존하는 프로쎄스에서 새로운 스레드를 창조하는것은 새로운 프로쎄스를 창조하는것보다 적은 작업량이 포함되며 (2) 같은 프로쎄스에서 스레드들사이의 통신이 간소화된다는것이다. 같은 프로쎄스에서 두 스레드사이의 방식절환이 서로 다른 프로쎄스에서 두 스레드사이에서의 방식절환보다 적은 작업을 포함한다는것도 역시 바로 그 경우로 되는가?
2. KLT에 비한 ULT의 설명에서 ULT의 결함은 ULT가 체계호출을 할 때 스레드가 폐색될뿐아니라 프로쎄스의 모든 스레드가 폐색된다는것이다. 그렇게 되는 리유는 무엇인가?

3. OS/2에서 다른 조작체계안에 있는 프로쎄스의 개념에서 공통적으로 구체화되는 것은 세개의 개별적인 실체들로 갈라 진다. 즉 전송단위, 프로쎄스와 스레드이다. 전송단위는 사용자대면부(건반, 영상표시장치, 마우스)와 관련된 하나이상의 프로쎄스들의 집합이다. 이 전송단위는 단어처리프로그람이나 작업판과 같은 대화형사용자응용프로그람을 표현한다. 이 개념은 개인용콤퓨터사용자가 여러개의 응용프로그람을 열고 화면상에 여러개의 매 창문들을 가지게 해준다.조작체계는 어느 창문인가에 대한 그리고 어느 전송단위가 능동인가에 대한 추적을 보존하여야 하며 그에 의해 건반과 마우스입력이 응용프로그람전송단위에로 경로를 정하게 된다. 임의의 시간에 하나의 전송단위는 전경방식에 있으며 다른 전송단위는 배경방식에 있다. 모든 건반과 마우스입력은 응용프로그람이 묘사하는데 따라 전경전송단위의 프로쎄스중의 하나에 주의를 돌리게 된다. 전송단위가 전경방식에 있을 때 영상출력을 수행하는 프로쎄스는 그것을 직접 하드웨어영상완충기에 보내며 거기로부터 사용자의 화면에로 출구된다. 전송단위가 배경방식으로 이동할 때 하드웨어영상완충기는 전송단위를 위한 론리영상완충기에 보관된다. 전송단위가 배경방식에 있는 동안 전송단위가 집행하고 화면출력을 산생하는 임의의 프로쎄스중의 스레드가 있다면 출력은 직접 론리영상완충기에로 경로를 정하게 된다. 전송단위가 전경방식으로 돌아갈 때 화면은 갱신되여 새로운 전경전송단위에서의 론리영상완충기의 현재 내용을 반영한다. OS/2에서 프로쎄스와 관련한 개념의 수를 세개로부터 둘로 감소시키는 방법이 있다. 전송단위를 없애고 사용자대면부(건반, 마우스, 화면)를 프로쎄스들과 련관시킨다.이렇게 하여 한번에 하나의 프로쎄스가 전경방식에 있다. 더 구조화하기 위해 프로쎄스들을 스레드로 쪼갤수 있다.

   ㄱ) 이 방법을 쓰면 어떤 편리성을 상실하는가?
   ㄴ) 이런 변경을 진척시킨다면 어디에서 자원(기억기, 파일 등)을 할당하겠는가 즉 프로쎄스준위에서 하겠는가, 스레드준위에서 하겠는가?

4. 사용자준위스레드와 핵심부준위스레드사이에 1대 1사영이 있는 환경을 고찰하자. 이 환경은 프로쎄스에서 다른 스레드가 계속 실행하는 동안 하나이상의 스레드를 체계호출차단시키도록 해준다. 이 모형이 단일처리기기계상에서 그것들의 단일스레드식실행보다 다중스레드식프로그람실행을 더 빨리 할수 있게 하는 리유를 설명하시오.
5. 프로쎄스가 선취되고 아직 프로쎄스중의 스레드들이 실행중에 있다면 그것들을 계속 실행할수 있겠는가?
6. OS/390일반형콤퓨터조작체계는 주소공간과 과제들에 대한 개념들로 구조화되여 있다. 대략적으로 말하여 단일주소공간은 단일응용프로그람에 대응하며 다른 조작체계들에서 하나이하의 프로쎄스에 대응한다. 주소공간에서 많은 과제들을 발생시킬수 있으며 병행하여 집행할수 있는데 이것은 대체적으로 다중스레드처리개념에 대응한다. 이 과제구조를 관리하는데서 두개의 자료구조가 기본이다. 주소공간조종블로크(ASCB)는 OS/390에 필요한 주소공간이 집행중인가 아닌가 하는데 대한 정보를 담고 있다. ASCB안의 정보는 할당우선권, 주소공간에 배정된 실제 및 가상기억기, 주소공간에 있는 그만한 개수의 준비상태의 과제들 및 매개가 교체되여 나가는가 마는가에 대한 정보를 포함하고 있다. 과제블로크(TCB)

는 집행상태에 있는 사용자프로그람을 표현한다. 그것은 주소공간에서 과제를 관리하는데 필요한 정보를 담고 있으며 처리기의 상태정보, 과제의 일부분인 프로그람에 대한 지시기들 및 과제집행상태를 포함하고 있다. ASCB는 체계기억기에 보존되여 있는 전역구조이며 한편 TCB는 그것들의 주소공간에 보존되여 있는 국부구조이다. 조종정보를 전역 및 국부몫으로 쪼개는것이 가지는 우점은 무엇인가?

7. 8개의 처리기를 가진 다중처리기가 테프구동기를 20개 결합하였다. 체계에 복종된 규모가 큰 많은 일감들이 있는데 그 매개는 집행을 완료하는데 4개의 테프구동기를 요구한다. 매개 일감은 세개의 테프구동기만을 가지고 오랜기간 실행하며 후에 그 조작의 마감근방에서 짧은 기간동안 네번째 테프구동기를 요구한다고 하자. 또 그러한 일감은 끝없이 보장된다고 하자.

ㄱ) OS에서 일정작성프로그람이 네번째 테프구동기를 사용할수 없는한 일감을 시작하지 않는다고 하자. 일감이 시작될 때 네번째 구동기는 즉시에 할당되며 일감이 완수될 때까지 해방되지 않는다. 한번에 진척시킬수 있는 최대일감의 수는 몇인가? 이 방법을 따를 때 휴식상태에 남아 있을수 있는 최대최소의 테프구동기수는 얼마인가?

ㄴ) 테프구동기사용을 개선하여 같은 사간에 체계의 교착을 피하는 다른 대치방법을 제기하시오. 한번에 진척시길수 있는 최대일감의 수는 얼마인가, 휴식중에 있는 테프구동기수의 한계는 얼마인가?

8. Solaris의 ULT상태들에 대한 설명에서 ULT가 같은 우선권을 가진 다른 스레드에 양보할수 있다는것을 설명하였다. 보다 높은 우선권을 가진 실행가능한 스레드가 있을수 있으며 따라서 양보기능은 같은 층에서 같거나 보다 높은 우선권을 가진 스레드에 양보하는 결과를 가져 올수는 없는가?

제 5 장. 병행성 : 호상배제와 동기화

조작체계의 중심주제는 모두 프로쎄스 및 스레드관리와 관련되여 있다. 즉

- **다중프로그람처리** : 하나의 처리기에서 여러개의 프로쎄스관리
- **다중프로쎄스처리** : 다중처리기에서 여러개의 프로쎄스관리
- **분산처리** : 여러개의 분산된 콤퓨터체계에서 집행하는 여러개의 프로쎄스관리
  최근 클라스터의 보급은 이런 체계류형에 대한 첫 실례로 된다.

이 모든 령역의 기초이며 조작체계설계의 기초는 병행성이다. 병행성은 프로쎄스사이의 통신, 자원공유와 경쟁, 여러개 프로쎄스들의 동작상 동기, 프로쎄스에 대한 처리기시간의 할당을 포함하여 대다수의 설계문제들을 포함한다. 이 문제점들은 다중프로쎄스처리나 분산처리환경에서만이 아니라 지어 단일처리기식 다중프로그람처리체계에서도 생긴다.

병행성은 3개의 서로 다른 문맥에서 생긴다. 즉

- **다중응용프로그람**: 다중프로그람처리는 여러개의 능동적인 응용프로그람들사이에서 처리시간을 동적으로 공유하도록 하기 위해 발명하였다.
- **구조화응용프로그람**: 모듈설계 및 구조식프로그람작성원리의 확장으로서 일부 응용프로그람을 병행프로쎄스의 모임으로서 효과적으로 프로그람화할수 있다.
- **조작체계구조**: 같은 구조화의 우점이 체계프로그람작성자에게도 적용되며 조작체계자체가 프로쎄스나 스레드의 모임으로 실현된다.

이 문제의 중요성으로부터 이 책의 4개 장이 병행성과 관련된 문제점들에 중심을 두고 있다. 이 장과 다음 장에서는 다중프로그람처리 및 다중스레드처리체계에서의 병행성을 취급한다. 제13장, 제14장은 분산처리와 관련한 병행성문제점을 론의한다. 이 책의 나머지 부분에서 조작체계설계에서의 다른 중요한 문제점들을 론한다고 하여도 병행성은 다른 모든 문제점들에 대한 고찰에서 중요한 역할을 한다.

이 장은 병행성의 개념과 여러 병행프로쎄스의 집행에 대한 의미를 소개하는것으로부터 시작한다.[1] 우리는 병행프로쎄스를 지원하는데서 기본요구가 호상배제능력이라는것을 알고 있다. 즉 한개 프로쎄스가 그 능력을 허가 받는 동안 다른 프로쎄스를 동작과정으로부터 배제하는 능력이다. 이 장의 두번째 절에서는 호상배제를 실현하는 여러가지 방법을 론한다. 이것들은 모두 쏘프트웨어적인 풀이이며 바쁜기다림이라고 하는 수법을 사용해야 한다. 다음으로 호상배제를 지원할수 있는 하드웨어적수법도 고찰한다. 그다음 바쁜기다림을 포함하지 않으며 조작체계가 지원하든가 언어번역기가 시행할수 있는 풀이를 보게 된다. 또한 세가지 방법 즉 신호기, 감시기, 통보문넘기기를 고찰한다.

병행성에서 두개의 고전적인 문제는 개념을 설명하고 이 장에서 제시하는 방법들을 비교하는데 사용한다. 처음에 생산자/소비자문제를 받아 들여 실행실례로 사용한다. 이 장은 읽기자/쓰기자문제로 끝난다.

병행성에 대한 설명은 제6장에서 계속되며 그 장의 마감에 실례체계의 병행성수법에 대하여 설명하기로 한다.

---

[1]. 간단히 하기 위해 일반적으로 프로쎄스들의 병행집행이라고 한다. 사실상 앞의 장들에서 보았기때문에 일부 체계들에서 병행성의 기본단위는 프로쎄스라기보다 스레드이다.

제 1 절. 병행성의 원리

단일처리기식 다중프로그람처리체계에서 프로쎄스들은 시간적으로 교차처리되면서 동시에 집행된다(그림 2-12 ㄱ). 실제적인 병행처리가 진행되지 않는다고 하여도, 프로쎄스들사이에서 앞뒤절환에 간접소비시간이 포함되지 않는다고 하여도 교차처리식집행은 처리효률성과 프로그람구조화에서 주되는 리익을 준다. 다중처리기체계에서 프로쎄스들을 교차처리할수 있을뿐아니라 그것들을 중복시킬수 있다(그림 2-12 ㄴ).

얼핏 보기에 교차처리와 중복이 근본적으로 다른 집행방식을 표현하며 서로 다른 문제점들을 제기하는것처럼 보인다. 사실상 두 수법들은 모두 병행처리의 실례로 볼수 있으며 모두 같은 문제점들을 제기한다. 단일처리기의 경우 이 문제는 다중프로그람처리체계의 기본특성으로부터 나온다. 즉 프로쎄스의 상대적집행속도는 예측할수 없다는것이다. 그것은 다른 프로쎄스의 동작, 조작체계가 새치기를 조종하는 방법, 조작체계의 일정작성방법에 의존한다. 다음과 같은 난점들이 생긴다. 즉

1. 전역자원에 대한 공유가 위험을 동반한다. 실례로 두개의 프로쎄스가 같은 전역변수를 사용하고 둘다 그 변수에 대해 읽기쓰기를 하면 그때 여러가지 읽기쓰기를 집행하는 순서가 위험하다. 이 문제에 대한 실례를 다음의 분절에서 보여 준다.
2. 조작체계가 자원할당을 최량적으로 관리하기 어렵다. 실례로 프로쎄스 A가 특정한 입출력통로의 사용을 요청하여 조종을 넘겨 받은 다음 그 통로를 사용하기전에 중단될수도 있다. 조작체계가 통로를 단순히 페쇄하고 다른 프로쎄스가 그것을 사용하지 못하게 하는것은 기대할수 없다. 사실 이것은 제6장에서 설명하게 되는 교착조건을 일으킬수 있다.
3. 결과가 대체로 결정론적이 아니며 재생할수 없기때문에 프로그람적오유를 찾기가 매우 어려워 진다(실례로서 이 점에 대한 설명은 [LEBL 87], [CARR 89]를 보시오.).

앞에서 론의한 난점들은 다중처리기체계에서도 물론 나타나는데 그것은 여기서도 프로쎄스의 상대적인 집행속도를 예측할수 없기때문이다. 다중처리기체계는 또한 여러개 프로쎄스의 동시적집행에서 생기는 문제도 취급해야 한다. 그러나 근본적으로 그 문제들은 단일처리기체계에서의 문제들과 같다. 이것은 설명을 해나감에 따라 명백해 질것이다.

간단한 실례

다음의 수속을 고찰하자. 즉

```
void echo()
{
    chin = getchar( );
    chout = chin ;
    putchar (chout) ;
}
```

이 수속은 문자의 echo수속을 주는 프로그람의 기본요소들을 보여 주고 있는데 입력은 한번에 한개의 건반을 다칠 때마다 건반에서 얻어 진다. 매 입력문자는 변수 chin에 기억된다. 다음변수 chout에 이송되며 화면표시부에 보내진다. 임의의 프로그람이 이 수속을 반복호출하여 사용자의 입력을 받고 그것을 사용자화면에 현시할수 있다.

이제 단일사용자를 지원하는 단일처리기식 다중프로그람처리체계를 고찰하자. 사용자는 한 응용프로그람에서 다른 응용프로그람에로 이행할수 있으며 매개 응용프로그람은 같은 건반을 입력으로 그리고 같은 화면을 출력으로 사용할수 있다. 매개 응용프로그람이 *echo*수속을 사용할것을 요구하므로 그것이 모든 응용프로그람에 공용인 기억기의 일부분에 적재되여 있는 어떤 공유된 수속으로 되여 있는것처럼 보인다. 이렇게 단일한 *echo*수속복사만을 사용하므로 공간을 절약한다.

프로쎄스에서 주기억기를 공유하는것은 프로쎄스사이에서 효과적이며 밀접한 대화를 허용하는데 쓸모가 있다. 다음의 순차를 고찰해 보자. 즉

1. 프로쎄스 P1이 *echo*수속을 기동시키며 getchar기능이 끝난 다음 즉시에 새치기된다. 이 점에서 가장 최근에 입력된 문자 *x*가 변수 *chin*에 기억된다.
2. 프로쎄스 P2가 활성화되여 *echo*수속을 기동시키는데 이것이 마지막까지 실행되여 단일문자 *y*를 입력하고 화면에 표시한다.
3. 프로쎄스 P1을 계속 실행한다. 이때까지 값 *x*는 *chin*에 덧씌여 졌으며 따라서 상실된다. 대신 *chin*은 *y*를 담고 있는데 이것이 *chout*에 이송되여 표시된다.

이렇게 되여 첫 문자는 상실되고 두번째 문자가 두번 표시된다. 이 문제의 본질은 공유된 전역변수 chin에 있다. 여러개의 프로쎄스가 이 변수에 접근한다. 만일 하나의 프로쎄스가 전역변수를 갱신하고 다음에 중단된다면 또다른 프로쎄스는 첫 프로쎄스가 변수를 사용하기전에 그것을 변경시킬수 있다. 그러나 한번에 하나의 프로쎄스만이 그 수속안에 있을수 있다고 해보자. 그러면 앞에서 설명한 순차는 다음과 같은 결과를 낳게 된다. 즉

1. 프로쎄스 P1이 *echo*수속을 기동시키고 입력함수의 끝에서 즉시에 새치기된다. 이 점에서 가장 최근에 입력된 문자 *x*가 변수 chin에 기억된다.
2. 프로쎄스 P2가 활성화되며 *echo*수속을 기동시킨다. 그러나 P1이 현재 중단되여 있기는 하나 아직 *echo*수속안에 있으므로 P2는 수속을 입구하는데서 페색된다. 따라서 P2는 중단되여 *echo*수속의 사용가능성을 기다린다.
3. 시간이 좀 지나서 프로쎄스 P1이 재개되여 *echo*수속의 집행을 완료한다. 적당한 문자 *x*가 표시된다.
4. P1이 *echo*수속을 벗어날 때 이것은 P2에 대한 페색을 제거한다. P2가 후에 재개될 때 *echo*수속을 충분히 기동시킨다.

이 실례에서 배우게 되는것은 공유된 전역변수(및 공유된 전역자원)를 보호해야 한다는것과 그것을 하기 위한 유일한 방도는 변수에 접근하는 코드를 조종하는데 있다는것이다. 만일 한번에 하나의 프로쎄스만이 *echo*수속에 들어 갈수 있고 일단 *echo*수속안에서는 수속이 다른 프로쎄스에 사용되기전에 끝까지 실행되여야 한다는 규률을 세우면 방금 설명한 이 형태의 오유는 생기지 않는다. 그 규률을 어떻게 세우는가 하는것이 이 장의 기본문제로 된다.

단일처리기, 다중프로그람처리식조작체계가 있다고 가정하고 이 문제를 설명하였다. 실례는 단일처리기가 있을 때에도 병행성문제가 생긴다는것을 실증하고 있다. 다중처리기체계에서 보호된 공유자원에 대해 같은 문제가 생기며 풀이도 같다. 우선 공유식전역변수에 대한 접근을 조종하기 위한 아무러한 수법도 없다고 가정하자. 즉

1. 프로쎄스 P1과 P2는 둘다 집행중이고 개개로 개별처리기상에서 집행된다. 프로쎄스들은 둘다 *echo*수속을 기동시킨다.

2. 다음의 사건이 발생한다. 같은 행의 사건들은 병렬로 발생한다.

```
Process  P1                    Process  P2
.                              .
in = getchar( );               .
.                              in = getchar( );
chout = chin;                    chout = chin;
putchar (chout);               .
.                              putchar (chout);
.                              .
```

결과는 P1에서의 문자입력은 표시되기전에 없어 지고 P2에서의 문자입력은 P1과 P2 모두에 의하여 표시된다는것이다. 다시 한번에 하나의 프로쎄스만이 *echo*수속에 있을수 있다는 규률을 적용할 가능성을 추가하자. 그러면 다음의 순차가 발생한다. 즉

1. 프로쎄스 P1과 P2는 둘다 집행중이고 개개는 개별처리기상에서 집행된다. P1이 *echo*수속을 기동시킨다.
2. P1이 *echo*수속에 있는 동안 P2가 *echo*수속을 기동시킨다. P1이 여전히 *echo*수속 (P1은 중단되든지 또는 집행중이든지)에 있고 P2는 수속에 들어 가지 못하고 페색된다. 따라서 P2는 중단되여 *echo*수속의 사용가능성을 기다린다.
3. 후에 프로쎄스 P1이 *echo*수속의 집행을 끝내고 수속을 벗어 나며 집행을 계속한다. P1이 *echo*수속으로부터 벗어 나자마자 P2가 실행을 재개하여 *echo*수속을 집행하기 시작한다.

단일처리기체계에서 문제로 되는것은 새치기가 프로쎄스의 임의의 곳에서 명령집행을 정지시킬수 있다는것이다. 다중처리기체계인 경우에 조건은 같으며 더우기 두 프로쎄스가 동시에 집행중에 있을수 있고 량자가 같은 전역변수에 접근하려고 할수 있기때문에 문제가 생길수 있다. 그러나 두 형태의 문제에 대한 풀이는 같다. 즉 공유자원에 대한 접근을 조종하는것이다.

조작체계의 임무

병행성이 존재할 때 어떤 설계 및 관리문제가 제기되는가? 다음과 같은 임무들을 렬거할수 있다.

1. 조작체계는 여러가지 능동프로쎄스들에 대한 추적을 유지할수 있어야 한다. 이것은 프로쎄스조종블로크를 사용하여 수행되며 제4장에서 설명하였다.
2. 조작체계는 매 능동프로쎄스에 여러가지 자원들을 배정 및 해방하여야 한다. 이 자원들은 다음과 같은것들이다.
   - **처리기시간** : 이것은 제4편에서 설명하게 되는 일정작성기능이다.
   - **기억기** : 대부분의 조작체계가 가상기억기방안을 사용한다. 이 문제는 제3편에서 론의된다.
   - **파일** : 제12장에서 설명된다.
   - **입출력장치** : 제11장에서 설명된다.
3. 조작체계는 다른 프로쎄스가 간섭하지 못하도록 매개 프로쎄스의 자료와 물리적 자원을 보호해야 한다. 이것은 기억기, 파일, 입출력장치와 관련된 수법을 포함한다. 보호에 대한 일반적 취급은 제15장에서 한다.

4. 처리의 결과는 집행이 다른 병행프로쎄스의 속도에 관하여 상대적으로 진행되므로 그 속도에 무관계해야 한다. 이것이 이 장의 주제로 되여 있다.

속도독립성에 대한 문제점을 어떻게 설명할수 있겠는가 하는것을 리해하기 위해서는 프로쎄스가 대화할수 있는 방법들을 알아야 한다.

프로쎄스의 대화

프로쎄스들이 다른 프로쎄스의 존재를 알고 있는 정도에 기초하여 프로쎄스들이 대화하는 방법을 분류할수 있다. 표 5-1은 세가지 가능한 인식정도와 매개의 영향을 목록화하고 있다.

표 5-1. 프로쎄스의 대화

인식정도	관계	한 프로쎄스가 다른 프로쎄스에 주는 영향	잠재적인 조종문제
프로쎄스들이 모른다.	경쟁	• 한 프로쎄스의 결과가 다른 프로쎄스들의 동작에 무관계하다. • 프로쎄스의 시간맞추기가 영향을 받을수 있다.	• 호상배제 • 교착(다시 갱신할수 있는 자원) • 고갈
프로쎄스들이 간접적으로 서로 알고 있다.(실례로 공유된 객체)	공유에 의한 협동	• 한 프로쎄스의 결과가 다른 프로쎄스에서 얻은 정보에 의존할수 있다. • 프로쎄스의 시간맞추기가 영향을 받을수 있다.	• 호상배제 • 교착(다시 갱신할수 있는 자원) • 고갈 • 자료의 간섭
프로쎄스가 직접적으로 서로 알고 있다.(그것들사이에 사용할수 있는 통신기본지령을 가진다.)	통신에 의한 협동	• 한 프로쎄스의 결과가 다른 프로쎄스에서 얻은 정보에 의존할수 있다. • 프로쎄스의 시간맞추기가 영향을 받을수 있다.	• 교착(소비할수 있는 자원) • 고갈

- **프로쎄스들이 서로 모른다**: 이것은 함께 작업시키려고 하지 않는 독립적인 프로쎄스들이다. 이 환경에 대한 가장 좋은 실례는 여러개의 독립적인 프로쎄스들에 대한 다중프로그람처리이다. 이것은 일괄일감들이거나 대화형 전송단위든가 그 혼합일수 있다. 프로쎄스들이 함께 동작하지 않아도 조작체계는 자원을 위한 경쟁에 관심을 돌려야 한다. 실례로 두개의 독립적인 응용프로그람이 같은 디스크나 파일 또는 인쇄기에 접근하려고 할수 있다. 조작체계는 이러한 접근들을 조절해야 한다.
- **프로쎄스들이 서로 간접적으로 알고 있다**: 이것은 자기들의 개개의 프로쎄스 ID에 의해 입출력완충기와 같은 일부 객체들에 대한 접근을 공유하는 프로쎄스들이다. 그러한 프로쎄스들이 공통객체를 공유하는데서 협동동작을 한다.
- **프로쎄스들이 서로 직접 알고 있다**: 이것은 프로쎄스 ID에 의해 서로 통신할수 있으며 일부 동작에서 련합하여 작업할수 있도록 설계된 프로쎄스들이다. 그러한 프로쎄스들이 또한 협동동작을 한다.

조건들이 표 5-1에서 제기한것처럼 항상 명백히 분류되는것은 아니다. 오히려 여러개의 프로쎄스들이 경쟁과 협동의 두가지 측면을 가질수 있다. 그럼에도 불구하고 앞의 목록에서 세개 항목들 매개를 개별적으로 설명하고 조작체계에서의 련관관계를 판정하는것이 좋다.

자원을 위한 프로쎄스들사이의 경쟁

병행프로쎄스들은 같은 자원을 사용하려고 경쟁할 때 서로 충돌하게 된다. 그의 순수한 형식으로 다음과 같은 상황을 서술할수 있다. 둘이상의 프로쎄스들이 자기의 집행과정에 어떤 자원에 대한 접근을 요구한다. 매개 프로쎄스는 다른 프로쎄스의 존재를 모르고 있으며 매개는 다른 프로쎄스의 집행으로부터 영향을 받지 않는다. 이것으로부터 매개 프로쎄스가 그것이 사용하는 임의의 자원의 상태가 영향을 받지 않도록 할것이라는 결론이 나온다. 자원의 실례를 보면 입출력장치들, 기억기, 처리기시간 및 시계를 포함한다.

경쟁하는 프로쎄스사이에는 아무런 정보교환도 없다. 그러나 한 프로쎄스의 집행은 경쟁하는 프로쎄스의 동작에 영향을 줄수 있다. 특히 두개의 프로쎄스가 모두 단일한 자원에 접근하려고 한다면 하나의 프로쎄스가 조작체계에 의해 자원을 배정받지만 다른것은 기다려야 할것이다. 그러므로 접근을 무시당한 프로쎄스는 속도가 더디게 된다. 극단한 경우에 페색된 프로쎄스가 자원에 대한 접근을 전혀 할수 없어 충분히 결속하지 못하게 된다.

경쟁하는 프로쎄스의 경우에 필경 세가지 조종문제가 나선다. 우선 **호상배제**에 대한 요구이다. 둘이상의 프로쎄스가 공유할수 없는 단일한 자원 말하자면 인쇄기와 같은 자원에 접근할것을 요구한다고 하자. 집행과정에 매개 프로쎄스는 입출력장치에 지령을 보내고 상태정보를 받으며 자료를 송신하거나 수신하게 된다. 그러한 자원을 림계자원이라고 하며 그것을 사용하는 프로그람의 일부분을 프로그람의 림계구간이라고 한다. 중요한것은 한번에 하나의 프로그람만이 림계구간에 들어 올수 있다는것이다. 구체적 요구가 명백하지 않으므로 이 제한을 리해하고 시행하기 위해 단순히 조작체계에 의거할수는 없다. 인쇄기의 경우를 실례로 들면 개별적인 프로쎄스가 전체 파일을 인쇄하는 동안 인쇄기에 대한 조종을 가질것을 요구한다. 그렇게 하지 않으면 경쟁하는 프로쎄스들에 의해 동작이 혼란될것이다.

호상배제는 두개의 추가적인 조종문제를 낳는다. 그중의 하나가 **교착**이다. 실례로 두개의 프로쎄스 P1, P2 및 두개의 자원 R1과 R2를 고찰하자. 매개 프로쎄스가 자기의 기능을 수행하기 위해 두 자원에 모두 접근할것을 요구한다고 하자. 그러면 다음과 같은 상태가 있을수 있다. 즉 조작체계가 R1을 P2에 할당하고 R2를 P1에 할당한다. 매개 프로쎄스는 두 자원중의 하나를 기다리고 있다. 어느것도 다른 자원을 획득하고 두 자원을 요구하는 기능을 수행할 때까지 자기가 이미 차지한 자원을 놓아 주려고 하지 않는다. 두 프로쎄스는 교착된다.

마지막 조종문제는 **고갈**이다. 세개의 프로쎄스(P1, P2, P3)가 각각 자원 R에 주기적으로 접근하려고 한다고 하자. P1은 자원을 소유한 상태에 있고 P2와 P3은 둘 다 지연되여 그 자원을 기다리고 있다. P1이 림계구간을 벗어 나면 P2든가 P3이 R에 대한 접근을 허락받는다. 조작체계가 P3에 접근을 허락하고 P1은 다시 자기의 림계구간을 완료하기전에 접근을 요구한다고 하자. 조작체계가 P3이 끝난후에 P1에 접근을 허락하고 다음에 P1과 P3에 교대로 접근을 허락해 주면 P2는 교착상태가 없다고 할지라도 그 자원에 대한 접근을 영원히 무시당할수 있다.

경쟁에 대한 조종은 불가피하게 조작체계를 포함하는데 그것은 자원을 배정하는것이 바로 조작체계이기때문이다. 더우기 프로쎄스자체가 자원을 사용하기에 앞서 페쇄하는것

과 같이 일정한 방식으로 호상배제를 위한 요구를 표현할수 있는 능력을 요구한다. 어떠한 풀이든 조작체계로부터 페쇄기능의 제정과 같은 일정한 지원을 포함하게 될것이다. 그림 5-1은 추상적인 술어들로 호상배제수법을 설명하고 있다. 구조체 **parbegin** (P1, P2, ..., Pn)은 다음과 같은 의미를 가지고 있다. 즉 주프로그람의 집행을 중단시키고 수속 P1, P2, Pn의 병행집행을 기동하며 P1, P2, ..., Pn모두가 결속될 때 주프로그람을 다시 실행한다. 그림 5-1에서는 *n*개의 프로쎄스가 병행하여 집행된다. 매개 프로쎄스는 (1) ID가 옹근수인 일부 자원에서 동작하는 림계구간과 (2) 자원을 포함하지 않는 나머지구간을 가지고 있다. 호상배제를 위해 두가지 기능 즉 입구림계 및 출구림계기능이 주어 진다. 매개 기능은 경쟁하는 자원의 이름을 변수로 취한다. 같은 자원을 놓고 다른 프로쎄스가 그의 림계구간에 있는 동안 자기의 림계구간에 들어 가려고 하는 프로쎄스는 기다린다.

입구림계 및 출구림계기능을 주는 특정한 수법을 설명하는것이 남아 있다. 프로쎄스 대화의 다른 경우를 고찰하는 동안 이 문제를 잠시 뒤로 미루기로 한다.

공유에 의한 프로쎄스사이의 협동

공유에 의한 협동의 경우는 구체적으로는 모르면서 다른 프로쎄스들과 대화하는 프로쎄스들을 포괄하고 있다. 실례로 여러 프로쎄스들이 공유된 변수나 공유된 파일이나 자료기지에 접근할수도 있다. 프로쎄스들은 다른 프로쎄스들을 참조하지 않고 공유된 자원을 사용하거나 갱신할수 있지만 다른 프로쎄스가 같은 자료에 접근할수 있다는것은 알고 있다. 그러므로 프로쎄스들은 협동하여 그것들이 공유하는 자료를 합리적으로 관리할수 있게 해야 한다. 조종수법은 공유된 자료의 보존성을 담보해야 한다.

```
/*호상배제프로그람*/
const int n = /*프로쎄스의 수*/

void P( int i )
{
    while (true)
    {
            entercritical (i);
            /*림계구간*/;
            exitcritical (i);
            /*나머지*/;
    }
}
void main()
{
    parbeign (P(R1), P(R2), ..., P(Rn));
}
```

그림 5-1. 호상배제

자료가 자원들(장치, 기억기)에 유지되여 있으므로 호상배제, 교착 및 고갈에 대한 조종문제는 또 생긴다. 다만 차이는 자료항목들에 두가지 방식들인 읽기 및 쓰기방식으로 접근할수 있다는것이며 다만 쓰기동작은 호상 배타적이여야 한다는것이다.

그러나 이 문제들에서 새로운 요구 즉 자료간섭문제가 제기된다. 간단한 실례로 여러가지 자료항목들을 갱신할수 있는 업무프로그람을 고찰해 보자. 두개의 자료항목 a와 b가 관계 $a=b$로 유지되여 있다고 한자. 즉 하나의 값을 갱신하는 프로그람은 또 다른것을 갱신하여 그 관계를 유지해야 한다. 이제 다음의 두 프로세스를 고찰해 보자. 즉

```
P1 :
        a=a+1;
        b=b+1;
P2:
        b=2 * b;
        a=2 * a;
```

상태가 초기에 일치한다면 개별적으로 취한 매개 프로세스는 공유된 자료를 일치하는 상태에 놓을것이다. 이제 두 프로세스가 매개 개별적인 자료항목(a와 b)우에서 호상배제에 관계하는 다음의 병행집행을 고찰해 보자. 즉

$$a = a + 1;$$
$$b = 2 \times b;$$
$$b = b + 1;$$
$$a = 2 \times a;$$

이 집행순서마지막에 조건 $a=b$는 더이상 유지되지 않는다. 실례로 $a=b=1$인 상태에서 출발한다면 집행순서마지막에 $a=4$ 그리고 $b=3$이 된다. 매 프로세스에서의 전체적인 순차가 림계구간이 되도록 선언하여 이 문제를 피할수 있다.

이리하여 공유식으로 협동하는 경우에 림계구간에 대한 개념이 중요하다는것을 알수 있다. 이미 설명한(그림 5-1) 입구림계 및 출구림계기능에 대해 같은 추상적기능을 여기서 사용할수 있다. 이 경우에 기능에서의 인수는 변수, 파일 또는 어떤 다른 공유된 객체가 될수 있다. 더우기 자료의 보존성을 주기 위해 림계구간을 사용한다면 인수로 식별할수 있는 아무러한 특정의 자원이나 변수도 있을수 없다. 그 경우에 병행프로세스사이에 공유되여 있는 식별자가 분명 호상배제되여야 할 림계구간을 식별해 주는것으로 인수를 생각할수 있다.

통신에 의한 프로쎄스사이의 협동

설명한 첫 두 경우에 매 프로쎄스는 자기자체의 격리된 환경을 가지는데 그것은 다른 프로쎄스들을 포함하지 않는다. 프로쎄스사이의 대화는 간접적이다. 두 경우에 다 공유가 있다. 경쟁의 경우에 그것들은 다른 프로쎄스를 의식하지 않고 자원을 공유하고 있다. 둘째 경우에 그것들은 값들을 공유하고 있는데 매개 프로쎄스가 다른 프로쎄스를 구체적으로 의식하지 못해도 자료의 보존성유지에 대한 요구는 의식하고 있다. 그러나 프로쎄스들이 통신에 의해 협동할 때 여러가지 프로쎄스들은 모든 프로쎄스를 련결하는 공동의 노력으로 관여한다. 통신은 여러가지의 동작들을 동기화시키고 조정하기 위한 방법을 준다.

대체로 통신은 일정한 종류의 통보문들로 구성된다고 특징지을수 있다. 통보문을 송수신하는데서 기본지령은 프로그람작성언어의 한 부분으로 줄수도 있고 조작체계의 체계핵심부가 줄수도 있다.

통보문을 넘기는데서 프로쎄스들사이에 공유되는것이 아무것도 없으므로 호상배제가 이런 종류의 협동에서 조종요구로 되지 않는다. 그러나 교착 및 고갈문제가 생긴다. 교착

에 대한 실례로 두 프로쎄스가 페색되여 각자가 다른것으로부터의 통신을 기다릴수 있다. 고갈에 대한 실례로 다음의 동작을 하는 3개의 프로쎄스 P1, P2 및 P3을 고찰해 보자. P1은 P2든가 P3과 통신하려고 반복시도하고 있고 P2와 P3은 둘다 P1과 통신하려고 한다. P1과 P2는 정보를 반복하여 교환하고 한편 P3은 페색되여 P1로부터의 통신을 기다리는 어떤 절차가 일어 날수 있다. P1은 능동상태에 남아 있고 P3은 고갈되기때문에 교착은 전혀 없다.

호상배제에서의 요구

호상배제에 지원을 주는 기능이나 능력은 다음의 요구를 만족시켜야 한다.

1. 호상배제는 꼭 시행되여야 한다. 즉 같은 자원이나 공유된 객체에 대해 림계구간을 가지는 모든 프로쎄스들속에서 한번에 하나의 프로쎄스만이 자기의 림계구간에 들어 가게 된다.
2. 자기의 비림계구간에서 정지하는 프로쎄스는 다른 프로쎄스들과 간섭하지 말아야 한다.
3. 림계구간에 대한 접근을 요구하는 프로쎄스가 무한히 지연될 가능성은 없어야 한다. 즉 교착이나 고갈이 없어야 한다.
4. 그 어느 프로쎄스도 림계구간에 없을 때 자기의 림계구간에 대한 입장을 요청하는 프로쎄스는 지연없이 입장하도록 허용해야 한다.
5. 상대적인 프로쎄스속도나 처리기의 수에 대한 그 어떤 가정도 없어야 한다.
6. 프로쎄스는 유한한 시간동안만 자기의 림계구간에 남아 있는다.

호상배제에서의 요구를 만족시키는 몇가지 방법이 있다. 한가지 방법은 병행하여 집행할것을 바라는 프로쎄스들에 응답성을 부여하는것이다. 이렇게 하여 체계프로그람이든지 응용프로그람이든지 프로쎄스들이 프로그람언어나 조작체계로부터 아무런 지원도 받지 않고 서로 호상배제수행을 조정하도록 요구하게 된다. 이것은 쏘프트웨어적방법이라고 볼수 있다. 이 방법은 높은 처리부담과 오유를 가지지만 병행처리의 복잡성을 더 잘 리해하기 위한 방법을 모색하는데서는 쓸모가 있다. 이 문제는 제5장 제2절에서 설명한다. 두번째 방법은 특수목적의 기계명령을 사용하는것이다. 이것은 부담을 감소시키는 우점을 가지고 있으나 일반목적풀이로는 되지 못한다. 이것은 제5장 제3절에서 설명한다. 세번째 방법은 조작체계나 프로그람작성언어에 일정한 지원준위를 주는것으로 된다. 그러한 방법중에서 가장 중요한 세개를 제5장 제4절부터 제5장 제6절에서 설명한다.

제 2 절. 호상배제 : 쏘프트웨어적방법

쏘프트웨어적방법은 공유기억기를 가진 단일처리기나 다중처리기망에서 집행하는 병행프로쎄스들에서 실현할수 있다. 이 방법은 보통 기억기접근준위에서의 호상배제를 가정한다([LAMP 91]아니면 문제 10을 볼것). 즉 주기억기의 같은 위치에 대한 동시접근(읽기자 및/또는 쓰기자)은 접근허가순서가 미리 규정되여 있지 않아도 일정한 종류의 기억기중재자에 의해 직렬화된다. 이이상 하드웨어, 조작체계 및 프로그람작성언어에서의 그 어떤 지원도 가정하지 않는다.

Dekker의 알고리듬

딕스트라[DIJK65]는 네데를란드의 수학자 데커가 설계한 두 프로쎄스에서의 호상배제알고리듬을 보고하였다. 딕스트라의 방법대로 단계별로 풀이를 개발해 보자. 이 방법은 병행프로그람개발에서 맞다들리게 되는 많은 공통적인 오유를 례증하는 우점을 가진다.

첫번째 시도

이미 언급한바와 같이 호상배제는 하드웨어적으로 된 일정한 기초적인 배제수법에 의거해야 한다. 가장 일반적인것은 한번에 기억위치에 대한 하나의 접근만이 이루어질수 있다는 제한이다. 이 제한을 사용하여 표식자 turn을 붙인 전역기억기위치를 예약한다. 자기의 림계구간을 집행하려고 하는 프로쎄스(P0 또는 Pl)는 우선 turn의 내용을 검사한다. turn의 값이 프로쎄스의 번호와 같으면 프로쎄스는 자기의 림계구간에 들어갈수 있다. 그렇지 않으면 기다려야 한다. 기다리고있는 프로쎄스는 림계구간에 들어갈 때까지 turn의 값을 반복하여 읽는다. 이 수속을 바쁜기다림이라고 하는데 이것은 림계구간에 들어가도록 허락을 받을 때까지 아무것도 못한다는 의미에서 나온 말이다. 대신에 그것은 시간을 보내면서 변수를 주기적으로 검사해야 한다. 그러므로 그 기회를 기다리는 동안 처리기의 시간(차지)을 소비한다.

프로쎄스가 자기의 림계구간에 접근한후 그리고 그 구역의 집행을 끝낸후 다른 프로쎄스의 동작을 위해 turn의 값을 갱신해야 한다.

공식적인 용어로 공유화전역변수가 있다. 즉

```
int   turn = 0;
```

그림 5-2 ㄱ는 두개 프로쎄스에서의 프로그람을 보여 준다. 이 풀이는 호상배제의 기능을 담보하지만 두가지 약점을 가지고 있다. 우선 프로쎄스들이 림계구간사용에서 엄격히 교대해야 한다. 이리하여 집행속도는 두 프로쎄스중에서 보다 느린것에 따른다. P0이 자기의 림계구간을 시간당 한번만 사용하지만 Pl 이 시간당 1000번의 속도로 그의 림계구간을 사용하려고 한다면 P1은 P0과 보조를 맞추어야 한다. 훨씬 더 심각한 문제는 하나의 프로쎄스가 실패하면 다른 프로쎄스가 영원히 페색된다는것이다. 이것은 프로쎄스가 림계구간안에서 실패하든 그밖에서 실패하든 사실이다.

우와 같은 해석은 **협동루틴**에 대한 해석이다. 협동루틴은 그것들사이에서 집행조종을 앞뒤로 넘길수 있도록 설계되여 있다. 이것은 단일프로쎄스에서 유용한 구조화기술로는 되지만 병행처리에는 적합하지 않다.

두번째 시도

첫 시도에서의 문제는 실제상 량쪽 프로쎄스에 대한 상태정보를 요구할 때 림계구간에 들어 갈수 있는 프로쎄스의 이름을 기억하는것이다. 사실상 매개 프로쎄스는 하나가 실패하면 다른것이 여전히 그의 림계구간에 접근할수 있도록 그 림계구간에 대한 자기자체의 열쇠를 가진다. 이 요구를 만족시키기 위해 론리적벡토르인 기발 flag를 정의하여 flag [0]은 P0에 대응시키고 flag[1]은 P1에 대응시킨다. 매개 프로쎄스는 다른 프로쎄스의 기발을 검사할수 있지만 그것을 변경시킬수는 없다. 프로쎄스가 자기의 림계구간에 들어 가려고 할 때 다른 프로쎄스의 기발이 false값을 가질 때까지 주기적으로 검사하는데 이것은 다른 프로쎄스가 자기의 림계구간에 없다는것을 가리킨다. 프로쎄스는 즉시에 자기의 기발을 true로 설정하고 자기의 림계구간에 들어 간다. 자기의 림계구간을 떠날 때는 기발을 false로 설정한다.

공유화전역변수가 이제는 다음과 같이 된다.

```
enum        boolean {false = 0 ; true = 1 ; } ;
boolean     flag[2] = {false, false} ;
```

그림 5-2 ㄴ는 알고리듬을 보여 주고 있다. 한 프로쎄스가 기발설정코드를 포함하고 있는 림계구간밖에서 실패하면 그때 다른 프로쎄스는 페색되지 않는다. 사실 다른 프로

쎄스가 필요한만큼 자주 자기의 림계구간에 들어 갈수 있는데 그것은 다른 프로쎄스의 기발이 항상 false이기때문이다. 그러나 프로쎄스가 자기의 림계구간안에서 또는 자기의 림계구간에 들어 가기직전에 자기의 기발을 true로 설정한후에 실패하면 다른 프로쎄스는 영원히 페색된다.

ㄱ)

```
/* 프로쎄스 0 */
•
•
while (turn != 0)
/*아무것도 하지 않는다.*/
/*림계구간*/
turn = 1;
•
```

```
/* 프로쎄스 1 */
•
•
while (turn != 1)
/*아무것도 하지 않는다.*/
/*림계구간*/
turn = 0;
•
```

ㄴ)

```
/* 프로쎄스 0 */
•
•
while(flag[1])
/*아무것도 하지 않는다.*/
flag[0] = true;
/*림계구간*/
flag[0] = false;
•
```

```
/* 프로쎄스 1 */
•
•
while (flag[0])
/*아무것도 하지 않는다.*/
flag[1] = true;
/*림계구간*/
flag[1] = false;
•
```

ㄷ)

```
/* 프로쎄스 0 */
•
•
flag[0] = true;
while (flag[1])
/*아무것도 하지 않는다.*/
/*림계구간*/
flag[0] = false;
•
```

```
/* 프로쎄스 1 */
•
•
flag[1] = true;
while (flag[0])
/*아무것도 하지 않는다.*/
/*림계구간*/
flag[1] = false;
•
```

ㄹ)

```
/* 프로쎄스 0 */
•
•
flag[0] = true;
while (flag[1])
{
    flag[0]=false;
      /*지연 */
    flag[0] = true;
}
/*림계구간*/
flag[0] = false;
•
```

```
/* 프로쎄스 1 */
•
•
flag[1] = true;
while (flag[0])
{
    flag[1]=false;
      /*지연 */
    flag[1] = true;
}
/*림계구간*/
flag[1] = false;
•
```

그림 5-2. 호상배제시도

ㄱ- 첫번째 시도, ㄴ- 두번째 시도, ㄷ- 세번째 시도, ㄹ- 네번째 시도

이 풀이는 앞의 첫 시도보다는 못하다. 왜냐하면 그것은 호상배제조차 담보하지 못하기때문이다. 다음의 순차를 고찰해 보자.

P0은 while문을 집행하고 flag[1]이 false로 설정되였는가 본다.
P1은 while문을 집행하고 flag[0]이 false로 설정되였는가 본다.
P0은 flag[0]을 true로 설정하고 자기의 림계구간에 들어 간다.
P1은 flag[1]을 true로 설정하고 자기의 림계구간에 들어 간다.

두 프로쎄스가 모두 자기의 림계구간안에 있으므로 프로그람은 부정이다. 문제는 제안된 풀이가 상대적인 프로쎄스의 집행속도와 독립이 되지 못한다는것이다.

세번째 시도

프로쎄스는 다른 프로쎄스가 자기의 상태를 검사한후에 그러나 다른 프로쎄스가 림계구간에 들어 가기전에 자기의 상태를 변화시킬수 있으므로 두번째 시도는 실패하였다. 이 문제를 그림 5-2 ㄷ에서 보여 준바와 같이 단순히 두 명령문을 서로 바꾸는것으로 결정할수 있다.

앞에서와 같이 프로쎄스가 림계구간을 조종하는 기발설정코드를 가지고 있는 자기의 림계구간에서 실패한다면 다른 프로쎄스는 페색되며 한 프로쎄스가 자기의 림계구간밖에서 실패하면 다른 프로쎄스는 페색되지 않는다.

다음으로 프로쎄스 P0의 견지에서 호상배제가 담보되는가를 검사해 보자. 일단 P0이 flag[0]을 true로 설정했으면 P0이 자기의 림계구간에 들어 간후 나올 때까지 P1은 자기의 림계구간에 들어 갈수 없다. P0이 기발을 설정했을 때 P1은 이미 자기의 림계구간안에 있을수 있다. 그 경우에 P1이 림계구간을 떠날 때까지 while문에 의해 P0은 페색된다. P1에 대해서도 같다.

이것은 호상배제를 담보하지만 또 다른 문제도 낳는다. 만일 한 프로쎄스가 while문을 집행하기전에 두 프로쎄스가 모두 자기의 기발을 true로 설정하면 각자는 다른 프로쎄스가 자기의 림계구간안에 들어 갔다고 생각하므로 교착을 야기시킨다.

네번째 시도

세번째 시도에서 프로쎄스는 다른 프로쎄스의 상태를 모르고 자기의 상태를 설정한다. 매개 프로쎄스가 자기의 림계구간에 들어 갈것을 자기의 권한으로 주장할수 있기때문에 교착이 생긴다. 이 처지에서 빠져 나갈 기회는 없다. 매개 프로쎄스를 더 복종하게 만들어 주는 방법으로 이 문제를 해결할수 있다. 즉 매개 프로쎄스는 자기의 기발을 설정하여 자기의 림계구간에 들어 가려는 요구를 표명하지만 그림 5-2 ㄹ에서 보여 준바와 같이 다른 프로쎄스에 복종하기 위해 그 기발을 재설정할 준비를 한다.

이것은 정확한 풀이에 가깝지만 아직 완전하지는 못하다. 호상배제는 아직 세번째 시도에 대한 설명에서와 비슷한 리유를 써야 담보된다. 그러나 다음의 사건순차를 고찰해 보자.

P0은 flag[0]을 true로 설정한다.
P1은 flag[1]을 true로 설정한다.
P0은 flag[1]을 검사한다.
P1은 flag[0]을 검사한다.
P0은 flag[0]을 false로 설정한다.
P1은 flag[1]을 false로 설정한다.
P0은 flag[0]을 true로 설정한다.
P1은 flag[1]을 true로 설정한다.

이 순차는 불명확하게 확장될수 있으며 어느 프로쎄스도 자기의 림계구간에 들어 가지 못할수 있다. 엄격히 말해서 이것은 교착이 아니다. 왜냐하면 두 프로쎄스의 상대적 속도에서의 변화가 이 주기를 파괴하여 하나가 림계구간에 들어 가게 하기때문이다. 이 조건을 **생명페쇄**라고 한다. 프로쎄스의 모임이 자기의 림계구간에 들어 가려고 하지만 어느 프로쎄스도 성공할수 없을 때 교착이 생긴다. 생명페쇄에서는 성공하는 집행순서가 있을수 있지만 또한 그 어느 프로쎄스가 자기의 림계구간에도 들어 갈수 없는 하나 또는 그이상의 집행순차를 서술할수 있다.

방금 설명한 각본이 매우 오래동안 유지될상 싶지는 않다고 해도 가능한 각본으로는 된다. 그러므로 네번째 시도를 버린다.

정확한 풀이

량쪽 프로쎄스의 상태를 관찰할수 있어야 하는데 이것은 배렬변수기발이 보장한다. 그러나 네번째 시도가 보여 주는것처럼 이것이 충분하지는 못하다. 방금 고찰한《호상호의》문제를 피하자면 두 프로쎄스의 동작순서를 정해 주어야 한다. 첫번째 시도의 변수 true를 이 목적에 사용할수 있는데 이 경우에 변수는 어느 프로쎄스가 자기의 림계구간에 들어 갈수 있는가를 가리킨다.

이 풀이를 다음과 같이 서술할수 있다. P0이 자기의 림계구간에 들어 가려고 할 때 그것은 자기의 기발을 true로 설정한다. 다음 계속하여 P1의 기발을 검사한다. 그것이 false이면 P0은 즉시 자기의 림계구간에 들어 갈수 있다. 그렇지 않으면 P0이 turn을 찾아 본다. turn=0이라는것을 발견하면 그것은 자기 차례가 되였다는것을 알고 주기적으로 P1의 기발을 검사한다. P1은 일정한 시점에서 자기가 복종할 차례에 있다는것을 표기해 주며 자기의 기발을 false로 설정하고 P0이 집행하게 해 준다. P0이 자기의 림계구간을 사용한후에 기발을 false로 설정하여 림계구간을 해방하고 turn을 1로 설정하여 권한을 P1에 이송한다.

그림 5-3은 Dekker의 알고리듬에 대한 설명을 보여 주고 있다. Dekker의 알고리듬에 대한 증명은 련습으로 남겨 둔다(문제 6을 보시오.).

Peterson알고리듬

Dekker의 알고리듬은 호상배제문제를 해결하지만 리해하기 힘들고 정확성은 엄밀하게 증명되여 있는 좀 복잡한 프로그람이다. Peterson[PETE 81]은 간단하고 명백한 풀이를 내놓았다. 앞에서와 같이 전역배렬변수 *flag*는 호상배제에 관하여 매개 프로쎄스의 위치를 가리키며 전역변수 *turn*은 동시성모순을 해결한다. 이 알고리듬을 그림 5-4에 보여주었다. 호상배제가 보존되는것을 쉽게 보여 주고 있다. 프로쎄스 P0을 고찰해 보자. 일단 그것이 *flag*[0]을 true로 설정했으면 P1은 자기의 림계구간에 들어 갈수 없다.

P1이 이미 자기의 림계구간안에 있으면 *flag*[1]=true로 되고 P0은 자기 림계구간에 들어 가지 못하고 페색된다. 다른 한편으로 호상페색이 보호된다. P0이 자기의 **while**고리에서 페색된다고 가정하자. 이것은 *flag*[1]이 true이고 *turn*=1이라는것을 의미한다. P0은 *flag*[1]이 false로 되든가 *turn*이 0으로 될 때 자기의 림계구간에 들어 갈수 있다. 이제 세가지 경우들을 고찰해 보자.

1. P1는 자기의 림계구간에 아무런 흥미도 없다. 이 경우는 불가능하다. 왜냐하면 그것은 *flag*[1]=false라는것을 의미하기때문이다.
2. P1이 자기의 림계구간을 기다리고 있다. 이 경우 또한 불가능하다. 그것은 *turn*

=1이면 P1이 자기의 림계구간에 들어 갈수 있기때문이다.

3. P1이 자기의 림계구간을 반복하여 사용하고 있고 따라서 그에 대한 접근을 차지하고 있다. 이런 일은 생길수 없다. 왜냐하면 P1이 매번 자기의 림계구간에 들어 가려고 하기전에 P0에 *turn*을 0으로 설정할 기회를 주기때문이다.

그러므로 두 프로쎄스의 호상배제에 대한 간단한 풀이를 가지게 된다. 더우기 Peterson의 알고리듬은 *n*개 프로쎄스의 경우에로 쉽게 일반화된다[HOFR 90].

제 3 절. 호상배제 : 하드웨어적지원

새치기금지

단일처리기기계에서 병행프로쎄스들은 중복될수 없고 다만 교차처리될수 있다. 더우기 한 프로쎄스가 조작체계봉사를 기동할 때까지 또는 새치기될 때까지 계속 실행된다. 따라서 호상배제를 담보하기 위해서는 프로쎄스가 새치기되지 못하게 보호하면 충분하다. 이 능력은 체계핵심부가 새치기를 금지 및 허락할수 있도록 정의된 기본지령의 형태로 실현할수 있다. 프로쎄스는 다음의 방법으로 호상배제를 실현할수 있다(그림 5-1과 비교). 즉

```
while ( turn )
{
  /*새치기금지*/;
  /*림계구간*/;
  /*새치기허락*/;
  /*나머지*/;
}
```

림계구간이 새치기될수 없으므로 호상배제는 담보된다. 그러나 이 방법은 비용이 많이 든다. 집행효률이 현저하게 떨어 진다. 그것은 처리기가 프로그람들을 교차처리하는 능력이 제한되기때문이다. 두번째 문제는 이 방법이 다중처리기구성방식에서 동작하지 않는다는것이다. 콤퓨터체계가 하나이상의 처리기를 가질 때 하나이상의 프로쎄스를 한번에 집행할수 있다. 이 경우에 금지된 새치기들은 호상배제를 담보하지 않는다.

특수기계명령

다중처리기구성에서 몇개의 처리기가 공통주기억기에 대한 접근을 공유한다. 이 경우에 주인/종속관계는 없고 오히려 처리기들은 동등한 관계에서 독립적으로 동작한다. 호상배제를 기초로 할수 있는 처리기들사이의 그 어떤 새치기수법도 없다.

하드웨어준위에서는 언급된바와 같이 어떤 기억위치에 대한 접근이 같은 위치에 대한 임의의 다른 접근을 배제한다. 이것을 기초로 하여 처리기설계자들은 하나의 명령불러내기주기로서 단일한 기억기위치에 대한 읽기 및 쓰기 또는 읽기 및 검사와 같은 두 동작을 하나로 수행하는 몇개의 기계명령을 제안하였다. 이 동작은 단일명령주기내에 수행되므로 다른 명령으로부터 간섭을 받지 않는다.

이 절에서는 가장 일반적으로 실현된 명령들중에서 두개를 본다. 다른것들은 [RAYN 86]과 [STON 93]에 서술되여 있다.

```
boolean flag [2];
    Int turn;
    Void P0( )
      {
      while ( true )
    {
  flag [0] = true ;
         while (flag [1] )
               if ( turn = = 1 )
               {
                  flag [0] = flase;
                     while ( turn = = 1 )
                     /*아무것도 하지 않는다.*/;
               flag [0] = true;
               }
        /*림계구간*/;
        turn = 1;
        flag [0] = false;
        /*나머지*/;
      }
    }
    void P1( )
    {
       while ( true )
       {
          flag [1] = true;
          while ( flag [0] )
             if ( turn = = 0 )
             {
                flag [1] = flase;
                while ( turn = = 0 )
                /*아무것도 하지 않는다.*/;
                flag [1] = true;
             }
          /*림계구간*/;
          turn = 0;
          flag[1] = false;
          /*나머지*/;
       }
    }
    void main ( )
    {
       flag [0] = false;
       flag [1 ]= false;
       turn = 1;
```

그림 5-3. 데커의 알고리듬

```
boolean flag [2];
  Int turn;
  void P0 ( )
    {
    while ( true )
  {
        flag [0] = true;
        turn = 1;
      while ( flag [1] && turn = = 1 )
          /*아무것도 하지 않는다.*/;
      /*림계구간*/;
      flag [0 ]= false;
      /*나머지*/;
    }
  }
  void P1 ( )
    {
    while ( true )
    {
        flag [1] = true;
        turn = 0;
        while ( flag [0] && turn = = 0 )
              /*아무것도 하지 않는다,*/;
        /*림계구간*/;
        flag [1] = false;
        /*나머지*/
     }
    }
   void main()
     {
      flag [0] = false;
      flag [1] = false;
      parbegin ( P0, P1 );
   }



       parbegin ( P0, P1 );

}
```

그림 5-4. 두 프로세스에서 피터슨의 알고리듬

검사 및 설정명령

검사 및 설정명령을 다음과 같이 정의할수 있다. 즉

```
boolean testset (int I)
 {
   if ( i == 0)
     {
       i = 1;
       return true;
     }
     else
     {
       return false;
     }
 }
```

```
/*호상배제프로그람*/
const int n = /*프로세스의 수*/;
int bolt;
void P(int I)
{
     while (true )
     {
           while(!testset(bolt))
              /*아무것도 하지 않는다.*/;
           /*림계구간*/;
           bolt=0;
       /*나머지*/}
}
void main()
{
     bolt=0;
     parbegin (P(1), P(2),..., P(n));
}
```

ㄱ)

```
/*호상배제프로그람*/
const int n = /*프로세스의 수*/;
int bolt;
void P(int i)
{
     int keyi;
     while (true)
     {
           keyi=1;
     while(keyi!=0)

           exchange(keyi, bolt)
           /*림계구간*/;
           exchange(keyi,bolt);
           /*나머지*/
        }
}
void main()
{
     bolt=0;
     parbegin (P(1), P(2),..., P(n));
}
```

ㄴ)

그림 5-5. 호상배제에서의 하드웨어적지원: ㄱ-검사 및 설정명령, ㄴ-교환명령

이 명령은 자기의 인수 *i* 값을 검사한다. 그 값이 0이면 그것을 1로 교체하고 true를 돌려 준다. 그렇지 않으면 그 값을 변화시키지 않고 false를 돌려 준다. 총적인 검사설정기능은 자동적으로 수행된다. 즉 새치기의 영향을 받지 않는다.

그림 5-5 ㄱ는 이 명령을 사용하는 호상배제규약을 보여 주고 있다. 공유화변수 *bolt*는 0으로 초기화된다. 자기의 림계구간에 들어 갈수 있는 프로세스만이 *bolt*가 0파

같다. 자기의 림계구간에 들어 가려고 하는 다른 모든 프로쎄스들은 기다림방식에 들어간다. 프로쎄스가 자기의 림계구간을 떠날 때 그것은 *bolt*를 0으로 재설정한다. 이 시점에서 기다리고 있는 프로쎄스하나만이 자기의 림계구간에 대한 접근을 넘겨 받는다. 프로쎄스의 선택은 어느 프로쎄스가 검사설정명령을 다음에 집행하게 되는가 하는데 관계된다.

교환명령

교환명령을 다음과 같이 정의 할수 있다. 즉

```
void exchange ( int register, int memory )
 {
     int temp;
     temp = memory;
     memory = register;
     register = temp;
 }
```

이 명령은 등록기의 내용을 기억위치의 내용과 교환한다. 이 명령집행기간에 기억위치에 대한 접근은 그 위치를 참조하는 임의의 다른 명령에 대하여 폐색된다.

그림 5-5 ㄴ는 이 명령을 사용하는 호상배제규약을 보여 주고 있다. 공유화변수 *bolt*는 0으로 초기화되여 있다. 매개 프로쎄스는 1로 초기화되여 있는 국부변수 *key*를 사용한다. 자기의 림계구간에 들어 갈수 있는 프로쎄스만이 *bolt*가 0과 갈다는것을 알수 있다. 그것은 *bolt*를 1로 설정하여 모든 다른 프로쎄스를 림계구간으로부터 배제한다. 프로쎄스가 자기의 림계구간을 떠날 때 *bolt*를 0으로 재설정하면 다른 프로쎄스가 그의 림계구간에 대한 접근을 얻을수 있다.

다음의 식이 항상 성립한다는것을 지적해 둔다. 그것은 변수를 초기화하는 방법이 다르기때문이며 교환알고리듬의 성질이기때문이다. 즉

$$\text{bolt} + \sum_i \text{key}_i = n$$

bolt=0이면 아무 프로쎄스도 자기의 림계구간안에 없다. *bolt*=1이면 정확히 하나의 프로쎄스가 자기의 림계구간안에 있다. 즉 *key*값이 0과 같은 프로쎄스가 그안에 있다.

기계명령법의 특성

특수한 기계명령을 사용하여 호상배제를 실현하는것은 여러가지 우점을 가지고 있다.

- 주기억기를 공유하고 있는 단일처리기나 다중처리기는 그 상에서 동작하는 임의의 수의 프로쎄스에 적용할수 있다.
- 단순하며 따라서 검증하기 쉽다.
- 여러 림계구간을 지원하는데 사용할수 있다. 매개 림계구간은 자기자체의 변수로 정의할수 있다.

일부 심중한 결함들도 있다.

- **바쁜기다림을 사용한다.** 그러므로 프로쎄스가 림계구간에 대한 접근을 기다리고 있는 동안 처리기시간을 계속 소비한다.
- **고갈이 생길수 있다.** 프로쎄스가 림계구간을 벗어 나고 하나이상의 프로쎄스가 기다리고 있을 때 기다리고 있는 프로쎄스를 임의로 선택할수 있다. 그러므로

일부 프로쎄스가 접근을 무시당할수 있다.

- **교착이 생길수 있다.** 단일처리기체계에 대한 다음과 같은 방안을 고찰해 보자. 프로쎄스 P1이 특수명령(실례로 검사설정, 교환)을 집행하고 자기의 림계구간에 들어 간다. P1은 그다음 새치기되여 보다 높은 우선권을 가진 P2에 처리기를 준다. 이제 P2가 P1과 같은 자원을 사용하려고 한다면 호상배제수법때문에 접근을 무시당한다. 그래서 그것은 바쁜기다림고리에 들어 간다. 그러나 P1은 다른 준비상태의 프로쎄스 P2보다 낮은 우선권을 가지고 있으므로 절대로 배분되지 않는다.

방금 개괄한 쏘프트웨어적 및 하드웨어적풀이는 모두 결함이 있기때문에 다른 수법을 찾아 보아야 한다.

제 4 절. 신호기

이제는 병행성을 보장하는데 사용되는 조작체계 및 프로그람작성언어수법을 보자. 이 절에서는 신호기로부터 시작한다. 다음 두개 절은 감시기와 통보문넘기기를 설명한다.

병행프로쎄스에 대한 문제를 취급하는데서 첫 주되는 전진은 1965년 Dijkstra의 론문으로부터 시작되였다[DIJK65]. Dijkstra는 협동하는 순차처리의 집합으로서 조작체계설계문제에 관심을 두었고 협동을 지원하는 효과적이며 믿음성 있는 수법을 개발하는데 관심을 두었다. 처리기와 조작체계가 그 수법들을 사용할수 있게 해준다면 사용자프로쎄스들이 이 수법들을 쉽게 사용할수 있다.

기본적인 원리는 다음과 같다. 즉 둘이상의 프로쎄스가 단순한 신호를 사용하여 협동할수 있다. 그 신호는 특정한 신호를 받을 때까지 지정된 위치에서 프로쎄스를 정지시킬수 있다. 복잡한 조정요구도 적당한 신호구조를 사용하여 만족시킬수 있다. 신호방식에서는 신호기라고 하는 특수한 변수를 사용한다. 신호기 *s*를 통해 신호를 전송하자면 프로쎄스가 기본지령 *signal*(*s*)를 집행한다. 신호기 *s*를 거쳐 신호를 수신하자면 프로쎄스는 기본지령 *wait*(*s*)를 집행한다. 만일 대응하는 신호가 아직 전송되지 않았다면 프로쎄스는 전송이 진행될 때까지 중단된다.[2]

요구하는 효과를 얻기 위해 신호기를 세가지 조작을 정의하는 기초우에서 옹근수값을 가지는 변수로 볼수 있다.

1. 신호기를 부아닌 값으로 초기화시킬수 있다.
2. *Wait*조작은 신호기값을 감소시킨다. 그 값이 부로 되면 *wait*를 집행하는 프로쎄스는 페색된다.
3. *Signal*조작은 신호기값을 증가시킨다. 그 값이 정이 아니면 *wait*조작으로 페색된 프로쎄스가 페색에서 벗어 난다.

이 세가지 조작을 내놓고 신호기를 검사하거나 조작할 방법은 없다.

그림 5-6은 신호기에서의 기본지령에 대한 보다 공식적인 정의를 주고 있다. *wait*와 *signal*의 기본지령에는 원자핵과 같은 가정을 주고 있다. 즉 그것들을 새치기할수 없고 매개 루틴들을 분해할수 없는 단계들로 취급할수 있다. **2진신호기**라고 하는 보다 제한된 판본을 그림 5-7에서 정의하고 있다. 2진신호기는 값으로 0과 1만을 취할수 있다. 원리적으로 2진신호기를 실현하는것이 더 쉽고 그것은 일반신호기와 같은 표현능력을 가진다

[2] Dijkstra의 초기론문에서와 많은 문헌들에서 문자 P는 Wait용으로 그리고 문자 V는 signal용으로 사용하고 있다. 이것들은 네데를란드말인 시험(proberen)과 증가(verhogen)의 첫 글자들이다.

(문제 13을 보시오.).

신호기와 2진신호기모두에 대하여 신호기를 기다리는 프로쎄스들을 유지하기 위한 대기렬을 사용한다. 프로쎄스를 그러한 대기렬에서 제거하는 순서에는 문제점이 있다. 가장 공평한 방법은 선입선출(FIFO)이다. 즉 가장 오래 폐색되여 있는 프로쎄스가 먼저 대기렬에서 해방된다. 이 방법을 정의하고 있는 신호기를 **강한 신호기**라고 부른다. 프로쎄스를 대기렬에서 제거하는 순서를 지정하지 않는 신호기를 **약한 신호기**라고 한다. 그림 5-8은 [DENN 84]에 기초한것으로서 강한 신호기의 조작에 대한 실례이다. 여기서 프로쎄스 A, B 및 C는 프로쎄스 D로부터의 결과에 의존한다. 초기에 (1) A는 실행중이고 B, C 및 D는 준비상태이며 신호기의 계수는 1로서 D의 결과중의 하나는 사용할수 있다는것을 가리킨다. A가 *wait*명령을 내보낼 때 그것은 즉시에 신호기를 넘겨 주고 집행을 계속할수 있다. 그다음에 그것은 준비대기렬을 재결합시킨다. 다음 B는 (2)를 실행하고 결국 *wait*명령을 내보내고 중단되여 D가 (3)을 실행한다. D가 새로운 결과를 완성하면 *signal*명령을 내보내는데 이것은 B가 준비대기렬 (4)에 이동하도록 한다. D가 준비대기렬을 재결합시키고 C가 (5)를 실행하기 시작하지만 *wait*명령을 내보낼 때 중단된다. 류사하게 A와 B가 실행하고 신호기에 따라 중단되며 D가 집행(6)을 재개하도록 한다. D가 결과를 가질 때 신호를 내보내는데 이것은 C를 준비목록에로 이송시킨다. D의 다음주기에는 A와 B를 중단에서 해방시킨다.

```
struct semaphore {
    int count;
    queueType queue;
}

void wait(semaphore s)
{
    s.count—;
    if (s.count < 0) f
    {
          프로쎄스를 s.queue에 놓
          고 폐색시킨다.
    }
}
void signal(semaphore s)
{
    s.count++;
    if (s.count <= 0)
    {
        프로쎄스  p를 s.queue에서
        꺼내고 준비목록에 놓는다.
    }
}
```

그림 5-6. 신호기의 기본지령에 대한 정의

```
struct binary _semaphore {
      enum (zero, one) value;
      queueType queue;
};

void waitB(binary _semaphore s)
{
    if (s.value == 1)
        s.value = 0;
    else
      {
         프로쎄스를 s.queue에 놓
         고 폐색시킨다.
      }
}
void signalB(semaphore s)
{
    if (s.queue.is_empty())
        s.value =1;
    else
    {
       프로쎄스  p를 s.queue에서
       꺼내고 준비목록에 놓는다.
    }
}
```

그림 5-7. 2진신호기의 기본지령의 정의

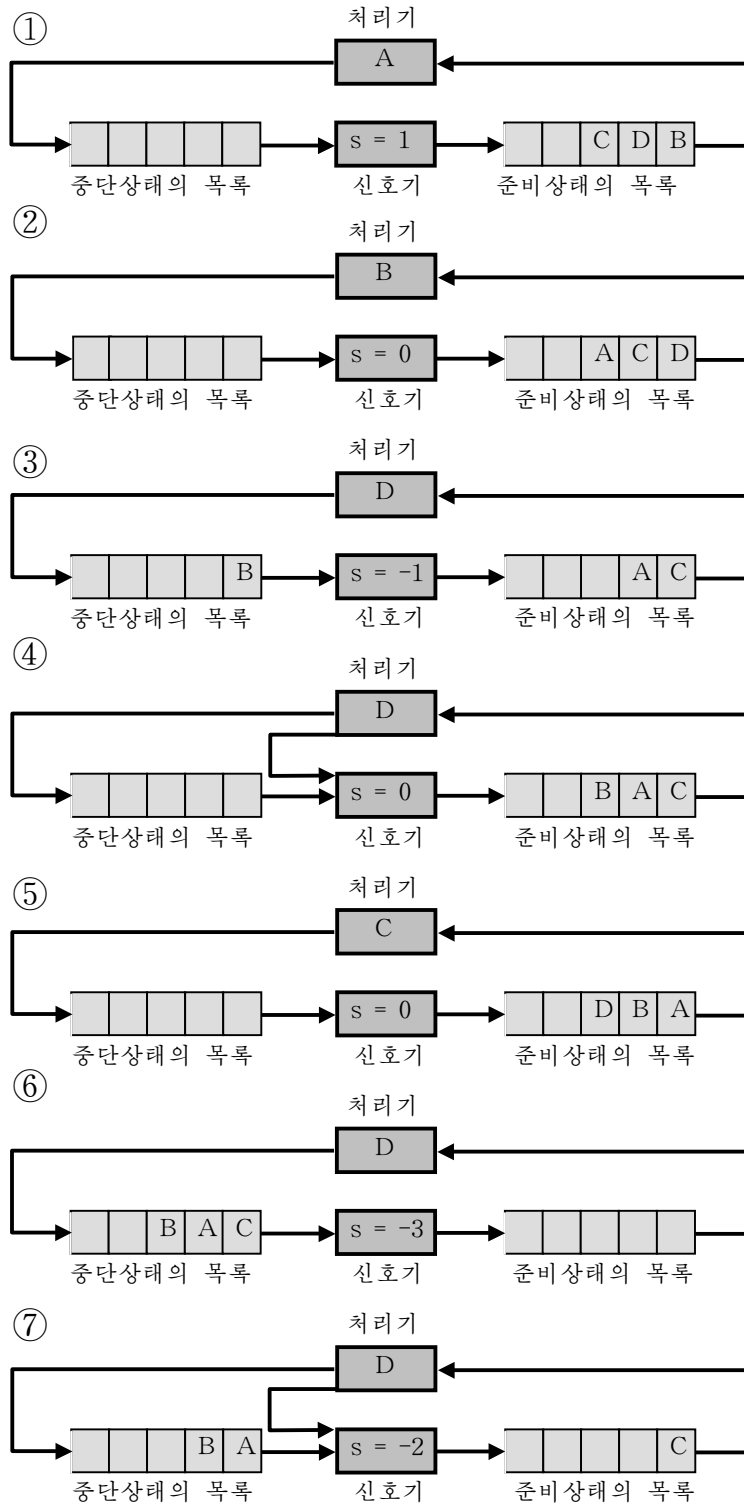



그림 5-8. 신호기기구의 실례

다음절에서 론의되며 그림 5-9에서 설명하고 있는 호상배제알고리듬에 대하여 강한 신호기는 고갈에 대처할수 있지만 약한 신호기는 그렇지 못하다. 강한 신호기가 더 편리하고 조작체계가 제공하는 전형적인 신호기형식이므로 보통 신호기를 강한 신호기로 한다.

호상배제

그림 5-9는 신호기 *s*를 사용하는 호상배제문제에 대한 간단한 풀이를 보여 주고 있다(그림 5-1과 비교하시오.). 배렬 *P*(*i*)로 식별되는 *n*개의 프로쎄스가 있고 매 프로쎄스에서 자기의 림계구간직전에 *wait*(*s*)를 집행한다고 하자. *s*값이 부로 되면 프로쎄스는 중단된다. 값이 1이면 그것을 0으로 감소시키고 프로쎄스는 즉시에 자기의 림계구간에 들어 간다. *s*가 더이상 정이 아니므로 아무런 다른 프로쎄스도 자기의 림계구간에 들어 갈수 없다.

```
/*호상배제프로그람*/
const int n = /*프로쎄스의 수*/;
semaphore s = 1;
void P(int i)
{
    while (true )
    {
        wait(s);
         /*림계구간*/;
         signal(s);
         /*나머지*/
    }
}
void main()
{
    parbegin (P(1), P(2),..., P(n));
}
```

그림 5-9. 신호기를 사용하는 호상배제

신호기는 1로 초기화된다. 이렇게 하면 *wait*를 집행하는 첫 프로쎄스는 *s*값을 0으로 설정하면서 즉시에 림계구간에 들어 갈수 있다. 림계구간에 들어 가려고 하는 다른 프로쎄스는 그것이 차지중이라는것을 알게 되며 *s*값을 –1로 설정하면서 페색된다. 임의의 몇개의 프로쎄스가 입구를 시도할수 있는데 그런 성공할수 없는 시도들은 개개가 *s*의 값을 더 감소시키는 결과를 가져 온다. 초기에 자기의 림계구간에 들어 간 프로쎄스가 벗어 날 때 *s*는 증가되고 페색된 프로쎄스중의 하나(있다면)가 신호기와 관련된 페색된 프로쎄스의 대기렬에서 나와 준비상태에 들어 간다. 조작체계가 다음번 일정작성할 때 그것은 림계구간에 들어 갈수 있다.

그림 5-10은 그림 5-9의 호상배제규률을 사용하는 세개의 프로쎄스에서의 가능한 순차를 보여 주고 있다. 이 실례에서 세개의 프로쎄스(A, B, C)는 신호기 *lock*에 의해 보호된 공유화자원에 접근한다. 프로쎄스 A는 *wait*(*lock*)를 집행한다. 신호기가 기다림조작시에 값 1을 가지고 있으므로 A는 즉시에 자기의 림계구간에 들어 갈수 있으며 신

호기는 값 0을 취한다. A가 자기의 림계구간에 있는 동안 B와 C는 둘다 기다림조작을 수행하며 신호기의 사용가능성을 얻을 때까지 페색된다. A가 자기의 림계구간을 벗어나 *signal*(*lock*)를 수행할 때 대기렬안에서 첫 프로쎄스였던 B가 이제는 자기의 림계구간에 들어 갈수 있다.

그림 5-9의 프로그람은 자기의 림계구간에 한번에 하나이상의 프로쎄스를 허용할데 대한 요구를 잘 처리할수 있다. 이 요구는 신호기를 특정한 값으로 초기화시킴으로써 만족된다. 이렇게 하여 임의의 시간에 *s.count*의 값을 다음과 같이 해석할수 있다.

- *s.count*≥0 : *s.count*는 중단되지 않고 *wait*(*s*)를 집행할수 있는 프로쎄스의 수이다 [그동안에 아무런 *signal*(*s*)도 집행되지 않는다는 조건에서].
- *s.count*<0 : *s.count*의 크기는 *s.queue*안에서 중단된 프로쎄스의 수이다.

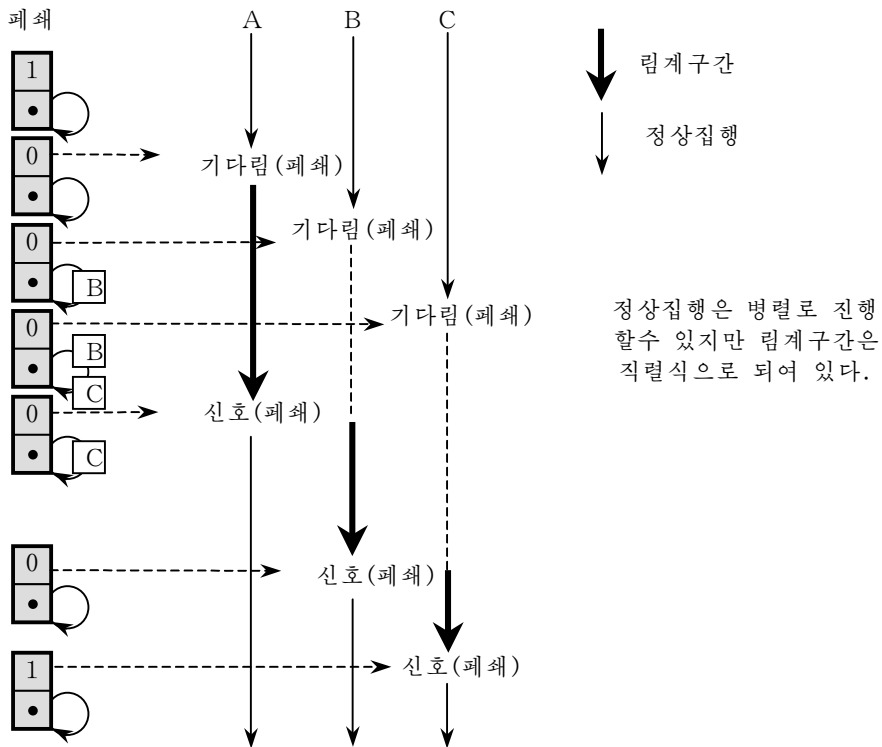



그림 5-10. 신호기에 의해 보호되는 공유자료에 접근하는 프로쎄스

생산자/소비자문제

병행처리에서 만나는 가장 일반적인 문제의 하나인 생산자/소비자문제를 보자. 일반적으로 이렇게 설명할수 있다. 즉 일정한 형태의 자료(레코드, 문자)를 발생시켜 완충기에 배치하는 하나이상의 생산자가 있다. 완충기에서 한번에 하나씩 항목을 꺼내는 단일한 소비자가 있다. 체계는 완충기조작의 중복을 막기 위해 제한시킨다. 즉 하나의 대리인(생산자 또는 소비자)만이 임의의 한순간에 완충기에 접근할수 있다. 이 문제에 대한 몇가지 풀이를 보고 신호기의 능력과 결함을 보기로 하자.

시작에 앞서 완충기는 무한이며 요소들이 선형배렬로 구성되여 있다고 가정하자. 추상적인 용어로 생산자와 소비자의 기능을 다음과 같이 정의할수 있다.

```
생산자:
while(true)
{
    /*생산자항목 v*/;
    b[in] = v;
    in ++;
}
```

```
소비자:
while(true)
{
    while(in <= out)
   /*아무것도 하지 않는다.*/;
w = b[out];
out++;
/*소비자항목 w*/
}
```

그림 5-11은 완충기 b의 구조를 보여 주고 있다. 생산자는 항목들을 발생하여 그것을 매 단계에서 완충기에 기억시킨다. 매번 완충기에 대한 첨수(*in*)가 증가된다. 소비자는 류사한 방식으로 진행하지만 빈 완충기로부터 읽으려고 하지 않도록 해야 한다. 이로부터 소비자는 생산자가 보다 앞섰다는것(*in* > *out*)을 확인해야 한다.

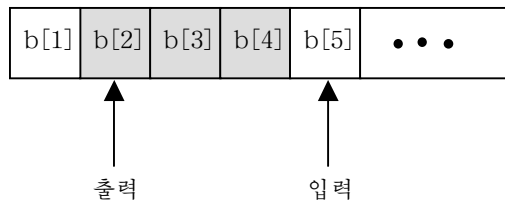



그림 5-11.생산자/소비자문제에서의 무한완충기

```
/*생산자소비자프로그람*/
int n;
binary_semaphore s = 1;
binary_semaphore delay = 0;
void producer()
{
    while(true)
    {
        produce();
        waitB(s);
        append();
        n++;
        if(n ==1)
            signalB(delay);
        signalB(s);
    }
}
```

```
void consumer()
{
int m; /*국부변수*/
  waitB(delay);
  while(true)
  {
     waitB(s);
     take();
     n--;
     signalB(s);
     consume();
     if(n= =0) waitB(delay);
  }
}
void main()
{
  n=0;
  parbegin(producer, consumer);
}
```

그림 5-12. 2진신호기를 사용하는 무한완충기방식의 생산자/소비자문제에 대한 부정확한 풀이

2진신호기로 이 체계를 실행해 보자. 그림 5-12가 첫 시도이다. 첨수 *in*과 *out*를 취급하기보다 옹근수변수 *n*(*n* = *in-out*)을 사용하여 완충기에서 몇개 항목들의 루적을 간단히 보존할수 있다. 신호기 *s*를 사용하여 호상배제를 수행한다. 신호기 *delay*는 완충기가 비여 있을 때 소비자가 기다리도록 하는데 사용한다.

이 풀이는 오히려 간단한것 같다. 생산자는 임의의 순간에 완충기에 자유롭게 추가할수 있다. 그것은 추가하기전에 *wait B*(*s*)를 수행하고 그다음에 *signalB*(*s*)를 수행하여 소비자 또는 다른 생산자가 추가조작기간에 그 완충기에 접근하지 못하게 막는다. 또한 자기의 림계구간에 있는 동안 생산자는 *n*의 값을 증가시킨다. *n* = 1이면 완충기는 이 추가직전에 비여 있었고 그래서 생산자는 *signal B*(*delay*)를 수행하여 이 사실을 소비자에게 알려 준다. 소비자는 *wait B*(*delay*)를 사용하여 첫 항목이 생산되기를 기다리기 시작한다. 다음항목을 취하고 자기의 림계구간에서 *n*을 감소시킨다. 생산자가 소비자의 앞에 있을수 있다면(일반적인 형편) *n*이 보통 정이므로 신호기 *delay*에서 좀처럼 페색되지 않을것이다. 이로부터 생산자와 소비자는 원활하게 실행한다.

그러나 이 프로그람에는 결함이 있다. 소비자가 완충기를 다 써버리면 *delay*신호기를 재설정할것을 요구하며 결국 생산자가 완충기에 항목을 더 배치할 때까지 기다려야 한다. 이것이 명령문 **if** *n* = = 0 *wait B*(*delay*)의 목적이다. 표 5-2에서 개괄한 방안을 고찰해 보자. 어두운 구역은 신호기 *s*가 조종하는 림계구간을 표시한다. 14행에서 소비자는 *waitB*조작을 집행하지 못한다. 소비자가 완충기를 실지로 다 소비했고 *n*을 0으로 설정하였지만(8행) 생산자는 소비자가 14행에서 그것을 검사하기전에 *s*을 증가시켰다. *signalB*가 앞의 *waitB*와 정합되지 못하는 결과가 생긴다. 20행에서 *n*의 값 –1은 소비자가 존재하지 않는 완충기에서 항목을 소비했다는것을 의미한다. 조건문을 소비자의 림계구간안으로 단순히 이동시킬수는 없다. 그것은 교착을 초래할수 있기때문이다(실례 8행이후).

표 5-2. 그림 5-11의 프로그램에서의 가능한 각본

	생산자	소비자	S	N	Delay
1			1	0	0
2	WaitB(s)		0	0	0
3	N++		0	1	0
4	If(n= =1)(signalB(lelay))		0	1	1
5	SignalB(s)		1	1	1
6		WaitB(delay)	1	1	0
7		WaitB(s)	0	1	0
8		n--	0	0	0
9		SignalB(s)	1	0	0
10	WaitB(s)		0	0	0
11	N++		0	1	0
12	If(n= =1)(signalB(lelay))		0	1	1
13	SignalB(s)		1	1	1
14		If(n= =1)(waitB(delay))	1	1	1
15		WaitB(s)	0	1	1
16		n--	0	0	1
17		SignalB(s)	1	0	1
18		If(n= =1)(waitB(delay))	1	0	0
19		WaitB(s)	0	0	0
20		n--	0	-1	0
21		SignalB(s)	1	-1	0

어두운 구역은 신호기가 조종하는 림계구간을 보여 준다.

```
/*생산자소비자프로그람*/
int n;
binary_semaphore s=1;
binary_semaphore delay=0;
void producer()
{
    while(true )
    {
        produce();
        waitB(s);
```

```
            append();
            n++;
            if(n = =1) signalB(delay);
            signalB(s);
        }
}
void consumer()
{
    int m; /*국부변수*/
    waitB(delay);
    while(true )
    {
        waitB(s);
        take();
        n--;
        m = n;
        signalB(s);
        consume();
        if(n = = 0) waitB(delay);
    }
}
void main()
{
    n=0;
    parbegin(producer, consumer);
 }
```

그림 5-13. 2진신호기를 사용하는 무한완충기방식의 생산자/소비자문제에 대한 정확한 풀이

이 문제에 대한 대답은 보조변수를 도입하는것으로서 그것을 소비자의 림계구간안에 설정하여 후에 사용할수 있게 하는것이다. 이것을 그림 5-13에서 보여 주고 있다. 론리적으로 주의 깊게 추적해 보면 교착이 더이상 생길수 없다는것을 확신할수 있다.

그림 5-14에서 보여 준바와 같이 일반적인 신호기(계수식신호기라고 부른다.)를 사용하면 보다 명확한 풀이를 얻을수 있다. 이 프로그람을 복사하는데서 오유를 범하여 조작 *signal*(*s*)와 *signal*(*n*)이 바뀌였다고 가정해 보자. 이것은 *signal*(*n*)조작이 소비자 또는 다른 생산자에 의한 새치기가 없이 생산자의 림계구간에서 수행될것을 요구한다. 이것이 프로그람에 영향을 주겠는가? 아니다. 왜냐하면 소비자가 임의의 경우 착수하기전에 두 신호기를 기다려야 하기때문이다.

이제 *wait*(*n*)와 *wait*(*s*)조작이 우연적으로 역전된다고 하자. 이것은 실로 심각하고도 치명적인 결함을 가져 온다. 완충기가 비였을 때 (*n.count* = 0) 소비자가 자기의 림계구간에 항상 들어 가면 그 어떤 생산자도 완충기에 영원히 추가할수 없으며 체계는 교착상태에 들어 간다. 이것은 신호기의 미묘성과 정확한 설계를 하는데서의 난점에 대한 좋은 실례로 된다.

```
/*생산자소비자프로그람*/
semaphore n=0;
semaphore s=1;
void producer()
{
   while(true )
   {
      produce();
      waitB(s);
      append();
      signal(s);
      signal(n);
   }
}
void consumer()
{
   while(true)
   {
      wait(n);
      wait(s);
      take();
      signal(s);
      consume();
   }
}
void main()
{
   parbegin(producer, consumer);
 }
```

그림 5-14. 신호기를 사용하는 무한완충기방식의 생산자/소비자문제에 대한 풀이

끝으로 생산자/소비자문제에 새롭고 실제적인 제한 즉 완충기가 유한하다는 제한을 주자. 완충기를 순환형기억기(그림 5-15)로 취급하며 지시기값들은 완충기의 크기를 법으로 하여 표현되여야 한다. 다음의 관계가 성립한다. 즉

폐색 :	**비폐색 :**
생산자: 충만된 완충기에 삽입한다.	소비자: 항목을 제거한다.
소비자: 빈 완충기에서 제거한다	생산자: 항목을 삽입한다.

생산자와 소비자의 기능을 아래와 같이 표현할수 있다(변수 *in*과 *out*는 0으로 초기화되여 있다).

그림 5-16은 일반신호기를 사용하는 풀이를 보여 주고 있다. 신호기 *e*는 빈 공간의 수를 추적하기 위하여 추가되였다.

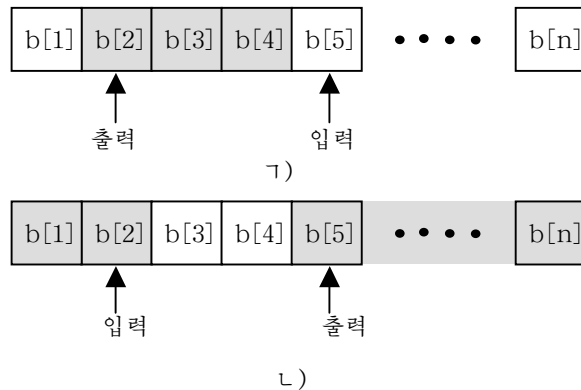



그림 5-15. 생산자/소비자문제에서의 유한순환형완충기

신호기의 실현

이미 언급된바와 같이 *wait*와 *signal*조작을 원자적인 기본지령들로 실현해야 한다. 하나의 명백한 방법은 그것을 하드웨어 또는 프로그람장치화수법으로 실현하는것이다. 이 방법을 내놓고도 여러가지 방안들이 제안되여 있다. 문제의 본질은 호상배제라는것이다. 즉 한번에 하나의 프로쎄스만이 *wait* 또는 *signal*조작으로 신호기를 조작할수 있다는것이다. 그러므로 Dekker의 알고리듬이나 Peterson의 알고리듬과 같은 쏘프트웨어방안을 사용할수 있는데 이것은 필수적인 처리의 간접소비시간을 요구한다. 또 다른 방법은 호상배제에 하드웨어의 지원을 받는 방안을 사용하는 방안들중에서 하나를 사용하는것이다. 실례로 그림 5-17 ㄱ는 검사 및 설정명령을 사용하는것을 보여 주고 있다. 이 실현에서 신호기는 역시 그림 5-9에서와 같은 구조이지만 새로운 옹근수성분 *s.flag*를 포함하고 있다. 명백히 이것은 바쁜기다림형식을 포함한다. 그러나 *wait* 및 *signal*조작은 상대적으로 짧으므로 포함되는 바쁜기다림의 량도 보잘것 없다.

```
/*경계완충기프로그람*/
 const int sizeofbuffer = /*완충기크기*/
 semaphore n = 0;
 semaphore s = 1;
 semaphore e = sizeofbuffer
 void producer()
 {
    while(true )
    {
       produce();
```

```
            wait(e);
            wait(s);
            append();
            signal(s);
             signal(n);
          }
    }
    void consumer()
    {
        while(true )
        {
            wait(n);
            wait(s);
            take();
            signal(s);
            signal(e);
            consume();
          }
    }
    void main()
    {
        parbegin(producer, consumer);
     }
```

그림 5-16. 신호기를 사용하는 경계완충기방식의 생산자/소비자문제에 대한 풀이

```
Wait(s)
{
   while (!estset(s.flag))
     /*아무것도 하지 않는다.*/;
   s.count--;
   if (s.count<0)
   {
       프로세스를 s.queue에 배치하고
       프로세스를 페색시킨다
       (또한 s.flag를 0으로 설정해야 한다.)
   }
     else
   s.flag=0;
}
signal(s)
{
   while (!estset(s.flag))
     /*아무것도 하지 않는다.*/;
   s.count++;
   if (s.count<=0)
   {
       프로세스 p를 s.queue에서 제거하고
       프로세스 p를 준비목록에 배치한다.
   }
  s.flag=0;
}
```

ㄱ)

```
Wait(s)
{
   inhibit interrupts;
   s.count--;
   if (s.count<0)
   {
         프로세스를 s.queue에 배치하고
         프로세스를 페색시킨다
         (또한 s.flag를 0으로 설정해야 한다.)
     }
     else
       새치기허용;
}
signal(s)
{
   inhibit interrupts;
   s.count++;
   if (s.count<=0)
   {
         프로세스 p를 s.queue에서 제거하고
         프로세스 p를 준비목록에 배치한다.
   }
   새치기허용;
}
```

ㄴ)

그림 5-17. 신호기의 두가지 가능한 실현: ㄱ-검사설정명령, ㄴ-새치기

단일처리기체계에서는 그림 5-17 ㄴ에 제기되여 있는것처럼 *wait*나 *signal*조작기간에 새치기를 금지시킬수 있다. 다시한번 강조하지만 이 조작기간이 상대적으로 짧다는것은 이 방법이 정당하다는것을 의미한다.

리발소문제

병행성을 실현하기 위해 신호기를 사용하는 또 다른 실례로서 간단한 리발소문제를 고찰한다.[3] 이 실례는 리발소자원들에 련이은 접근을 주려고 할 때 맞다들리는 문제가 실지 조작체계에서 맞다들리게 되는 문제와 류사하므로 론의할만하다.

리발소에는 3개의 리발의자, 세명의 리발사, 4명의 손님을 쏘파에 앉히고 다른 손님들은 서서 기다리는 방구역이 있다(그림 5-18). 파이어코드는 리발소에서 손님의 총수를 20명으로 제한한다. 이 실례에서는 리발소가 결국 50명의 손님들을 봉사하게 된다고 가정한다.

손님은 다른 손님들로 하여 수용능력이 다 차면 리발소에 들어 가지 못할것이다. 한편 내부에서는 손님이 쏘파에 자리를 잡거나 쏘파가 다 차 있으면 서 있는다. 리발사가 시간이 나면 쏘파에 가장 오래 있었던 손님이 봉사를 받게 되며 서 있는 손님이 있다면 리발소에 가장 오래 서 있은 손님이 쏘파에 자리잡게 된다. 손님의 리발이 끝날 때에 임의의 리발사가 료금을 받을수 있지만 출납대가 하나만 있으므로 료금은 한번에 한 손님에 대해서만 받는다. 리발사들은 머리를 깎고, 료금을 받으며, 손님을 기다리면서 자기의 리발의자에 앉아 있는데 시간을 할당하게 된다.

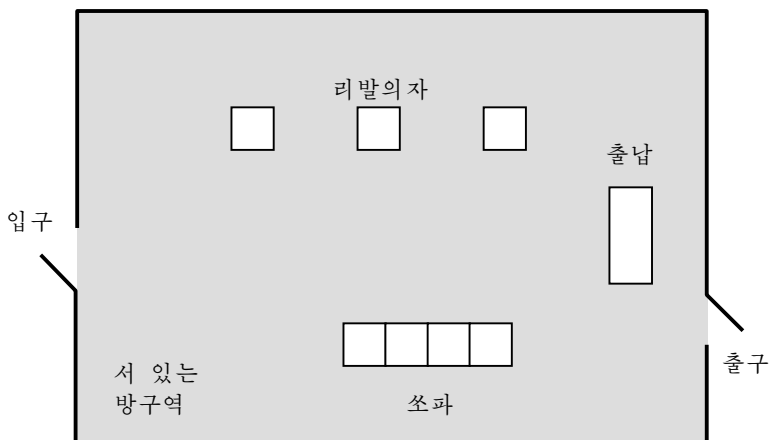



그림 5-18. 리발소

불공평한 리발소

그림 5-19는 신호기를 사용하는 실현을 보여 주고 있는데 공간을 절약하기 위하여 세개의 수속을 나란히 서술하고 있다. 모든 신호기의 대기렬이 선입선출방법으로 처리된다고 가정한다.

프로그람의 기본부분은 50명의 손님, 3명의 리발사와 출납처리를 활성화시킨다. 이제 여러가지 동기화조작기의 목적과 지위를 고찰해 보자. 즉

---

3. 이 문제는 Chico에 있는 캘리포니아 국가종합대학의 Ralph Hilzer 교수의 도움을 받아 필자가 내놓았다.

- **리발소와 쏘파:** 리발소의 능력과 쏘파의 능력은 각각 신호기 *max_capacity* 와 *sofa* 가 관리한다. 손님이 리발소에 들어 가려고 할 때마다 *max_capacity* 신호기는 1만큼 감소되며 손님이 나올 때마다 신호기는 증가된다. 손님이 리발소가 만원이라는것을 알면 그 손님의 처리는 *max_capacity* 상에서 *wait* 함수에 의해 중단된다. 류사하게 *wait* 와 *signal* 조작은 쏘파에 앉거나 일어 나는 동작들에 대응한다.
- **리발의자의 능력:** 3개의 리발의자가 있으며 그것을 합리적으로 사용하는데 주의를 돌려야 한다. 신호기 *barber_chair* 는 어떤 손님이 다른 사람의 무릎우에 올라 앉는 점잖지 못한 현상이 일어 나는것을 피하기 위해 한번에 세명이상의 손님이 봉사 받을수 없다는것을 담보해 준다. 손님은 적어도 한개의 의자가 빌 때에야 쏘파에서 일어 나게 되며 [*wait*(*barber_chair*)] 매 리발사는 손님이 자기 의자를 떠났을 때 신호를 한다. 리발의자에 대한 공평한 접근은 신호기의 대기렬구성이 제공한다. 즉 폐색되는 첫 손님이 의자를 사용할수 있는 첫 사람으로 된다. 손님의 수속에서 만약 *wait*(*barber_chair*)가 *signal*(*sofa*)후에 발생했다면 매 손님은 단지 쏘파에 잠간 앉은 다음 리발의자줄에 서게 되며 혼잡을 일으키고 팔꿈치가 부딪치는 비좁은 리발소를 떠난다.

```
/*리발소프로그람 1*/
semaphore max_capacity = 20;
semaphore sofa = 4;
semaphore barber_chair = 3;
semaphore coord = 3;
semaphore cust_ready = 0, finished = 0, leave_b_chair = 0, payment = 0, receipt = 0;
void customer()
{
wait(max(capacity);
enter_shop();
wait(sofa);
sit_on_sofa();
wait(barber_chair);
get_up_from_sofa();
signal(sofa);
sit_in_barber_chair;
signal(cust_ready);
wait(finished);
leave_barber_chair();
signal(leave_b_chair);
pay();
signal(payment);
wait(receipt);
exitshop();
signal(max_capacity)
}

Void barber()
{
while(true)
{
wait(cust_ready);
wait(coord);
cut_hair();
signal(coord);
        signal(finished);
    wait(leave_b_chair)
    ;
signal(barber_chair);
  }
}

Void cashier()
{
while(true)
{
wait(payment);
wait(coord);
accept_pay();
signal(coord);
signal(receipt);
   }

void main()
{
parbegin (customer, ... 50 times, ... customer, barber, barber, barber, cashier);

}
```

그림 5-19. 불공평한 리발소

- **손님이 리발의자에 있다는 확증:** 신호기는 자는 리발사의 깨우기신호로서 손님이 방금 의자를 차지했다는것을 가리킨다. 이 신호기가 없으면 리발사는 절대로 잘수 없고 손님이 의자를 떠나자마자 머리를 깎기 시작하게 될것이다. 만일 그 어떤 새 손님도 자리에 앉지 않았다면 리발사는 허공에 대고 리발을 하고 있을것이다.
- **손님을 리발의자에 앉히기:** 손님은 일단 앉았으면 리발사가 신호기 *finished* 를 사용하여 리발이 끝났다는 신호를 줄 때까지 그 의자에 앉아 있는다.
- **한 손님을 하나의 리발의자에 제한하기:** 신호기 *barber_chair*는 리발의자에 있는 손님수를 세명으로 제한한다. 그러나 *barber-chair*는 자체로 이것을 성공시키지 못한다. 리발사가 *signal(finished)*를 집행한후에 즉시 처리기를 얻지 못한 손님은(즉 다른 사람과 잡담하는데 정신이 팔려 있는 손님) 다음 손님이 앉을 허가를 받을 때까지 여전히 의자에 있을수 있다. 신호기 *leave_b_chair*는 꾸물거리는 사람이 자기가 떠난다는것을 알릴 때까지 리발사가 새 손님을 의자에 불러들이지 못하게 함으로써 이 문제를 정정할수 있다. 이 장의 마지막에 있는 문제에서는 이 예방책도 성급한 손님이 자리를 차지하는것을 막지 못한다는것을 알수 있다.
- **지불 및 령수증 받기:** 돈을 취급할 때는 자연히 주의를 돌리려고 한다. 출납은 매개 손님이 리발소를 떠나기전에 료금을 지불하는것을 확인하려고 하며 손님은 지불했다는 확증(령수증)을 받으려고 한다. 이것은 사실상 돈을 직접 넘겨 주어야 실현된다. 매 손님은 리발의자에서 일어 나자마자 료금을 지불하고 출납에 돈을 넘겨 주었다는것을 알리고 [*signal(payment)*]다음에 령수증을 기다린다 [*wait(receipt)*]. 출납프로쎄스는 반복하여 지불을 처리한다. 즉 지불이 신호를 보내기를 기다리며 돈을 받고 다음 돈을 받았다는 신호를 보낸다. 여기서 몇가지 프로그람작성오유를 피해야 한다. 만일 *signal(paymenet)*이 *pay*가 동작하기 직전에 발생했다면 손님은 신호를 보낸 다음 새치기될수 있는데 이것은 누구도 지불하지 않았다고 해도 출납이 지불을 받지 못하게 할수도 있다. 지어 더 심각한 오유는 *signal(payment)*과 *wait(receipt)*행들의 위치를 역전시키는것이다. 이것은 교착을 초래하게 되는데 그 리유는 그것이 모든 손님들과 출납이 자기들의 개별적인 *wait*조작에서 페색되게 하기때문이다.
- **리발사와 출납기능의 조정:** 돈을 절약하기 위해 리발소는 따로 출납을 쓰지 않는다. 매개 리발사는 머리를 깎는중이 아니면 그 일을 수행해야 한다. 신호기 *coord*는 리발사가 한번에 하나의 과제만을 수행할수 있도록 한다.

표 5-3 은 프로그람에서 매개 신호기의 사용에 대하여 요약하고 있다.

출납처리는 지불기능을 리발사수속에 합하여 제거할수 있다. 매개 리발사는 련속적으로 리발과 료금접수를 한다. 그러나 단일한 출납대에서 한번에 한 리발사에게 료금접수기능이 접근하도록 제한해야 한다. 그 기능을 림계구간으로 처리하고 신호기로 그것을 감시하여 이것을 수행할수도 있다.

표 5-3. 그림 5-18 에서 신호기들의 목적

신호기	기다림조작	신호조작
max-capacity	손님이 리발소에 들어 가기 위해 방을 기다린다.	나가는 손님이 기다리는 손님에게 나가라고 신호한다.
Sofa	손님이 쏘파에 앉아 기다린다.	쏘파를 떠나는 손님이 쏘파에 앉아 기다리는 손님에게 신호한다.

표계속

신호기	기다림조작	신호조작
barber-chair	손님이 빈 리발사의자를 기다린다.	리발사가 의자가 빌 때 신호한다.
cust-read	리발사가 손님이 의자에 들어 앉을 때까지 기다린다.	손님은 리발사에게 손님이 의자에 앉았다는것을 신호한다.
Ished	손님은 자기 리발이 될 때까지 기다린다.	리발사가 그 손님의 머리를 다 깎았을 때 신호한다.
leave-b-chair	리발사는 손님이 의자에서 일어날 때까지 기다린다.	손님이 출납에게 자기가 지불했다는것을 신호한다.
Payment	출납은 손님이 지불하기를 기다린다.	손님이 출납에게 자기가 지불했다는것을 신호한다.
Receipt	손님은 지불에 대한 령수증을 기다린다.	출납은 지불을 받았다는것을 신호한다.
Coord	리발사자원이 자유로워 져서 리발이든 출납기능이든 수행하기를 기다린다.	리발사자원이 자유롭다는것을 신호한다.

공평한 리발소

그림 5-19가 하나의 좋은 성과이기는 하지만 일부 난점도 있다. 이 절의 나머지 부분에서 한가지 문제를 해결하는데 다른것들은 독자들에게 련습으로 남겨 둔다(문제 19를 보시오.).

그림 5-19에는 손님들을 불공평하게 취급할수도 있는 시간맞물림문제가 있다. 현재 세 손님이 세개의 리발의자에 앉아 있다고 하자. 그 경우에 손님들은 분명 *wait*(*finished*)상에서 페색될것이며 대기렬구성으로 그들은 리발의자에 들어 간 순서로 나오게 된다. 그러나 한 리발사가 매우 빠르거나 손님중의 어떤 사람이 머리칼이 적다면 어떻게 되겠는가? 첫 손님이 의자에 들어 가 앉으면 한 손님이 자기가 앉은 의자에서 밀려 나 부분적리발에 대해 완전한 값을 물도록 하며 한편 다른 손님은 머리를 다 깎았음에도 의자를 뜨지 못하는 상태가 조성될수 있다.

그림 5-20에서 보여 준것처럼 신호기를 더 사용하여 이 문제를 해결한다. 매 손님에게 유일한 손님번호를 할당하는데 이것은 매 손님이 리발소에 들어 가는 번호를 가지는 것과 같다. 신호기 *mutex1*은 전역변수 *count*에 대한 접근을 보호하여 매 손님이 유일한 번호를 받도록 한다. 신호기 *finished*를 50개의 신호기배렬로 다시 정의한다. 일단 손님이 리발의자에 앉으면 그는 *wait*(*finished*[*custnr*])를 집행하여 자기자체의 유일한 신호기를 기다리게 하는데 리발사가 그 손님에 대한 봉사를 끝낼 때 리발사는 *signal*(*finished* [*b_cus t*])을 집행하여 정확히 그 손님을 내보낸다.

손님의 번호가 리발사에게 어떻게 알려 지게 하겠는가 하는 문제가 제기된다. 손님은 리발사에게 신호기 *cust-ready* 로 신호하기 직전에 대기렬 *enqueue1*에 자기번호를 놓는다. 리발사가 리발할 준비가 되면 *dequeue1*(*b_cust*)은 *queue1*에서 제일 우의 손님번호를 제거하고 그것을 리발사의 국부변수 *b_cust*에 놓는다.

```
/*리발소프로그람 2*/
semaphore max_capacity = 20;
semaphore sofa = 4;
semaphore barber_chair = 3, coord = 3;
semaphore mutex1= 1, mutex2 =1;
semaphore cust_ready = 0, finished = 0, leave_b_chair = 0, payment = 0,receipt = 0;
semaphore finished [50] = {0};
int count;

void customer()
{
  int custnr;
  wait(max_capacity);
  enter_shop();
  wait(mutex1);
  count++;
  custnr=count;
  signal(mutex 1);
  wait(sofa);
  sit_on_sofa();
  wait(barber_chair);
  get_up_from_sofa();
  signal(sofa);
  sit_in_barber_chair();
  wait(mutex 2);
  enqueuel(custnr);
  signal(cust_ready);
  signal(mutex2);
  wait(finished[custnr]);
  leave_barber_chair();
  signal(leave_b_chair);
  pay();
  signal(payment);
  wait(receipt);
  exit_shop();
  signal(max_capacity)
}

void barber()
{
  int b_cust;
  while(true)
  {
    wait(cust_ready);
    wait(mutex2);
    dequeue 1(b_cust);
    signal(mutex2);
    wait(coord);
    cut_hair();
    signal(coord);
    signal(finished[b_cust]
    wait(leave_b_chair);
    signal(barber_chair);
  }
}

void cashier()
{
  while(true)
  {
    wait(payment);
    wait(coord);
    accept_pay();
    signal(coord);
    signal(receipt);
  }
}

void main()
{  count:=0;
   parbegin (customer, ... 50 times, ... customer, barber, barber, barber, cashier);
}
```

그림 5-20. 공평한 리발소

제 5 절. 감시기

신호기는 호상배제에서와 프로쎄스조종에서 위력하고도 유연한 기본지령들을 준다. 그러나 그림 5-12에서와 같이 신호기를 사용하여 정확한 프로그람을 만들기는 힘들수 있다. 난점은 *wait*와 *signal*조작이 프로그람전반에 흩어 질수 있으며 그것들이 신호기상에서 이 조작들에 주는 총적영향을 예측하기가 쉽지 않다.

감시기는 신호기의 기능과 같은 기능을 가지고 있으며 더 쉽게 조종할수 있는 프로그람작성언어구조이다. 그 개념은 ［HOAR 74］에서 처음 공식적으로 정의되였다. 감시기구조는 Concurrent Pascal, Pascl-Plus, Modula-2, Modula-3 및 Java를 포함하여 많은 프로그람언어들에서 실현되여 있다. 그것은 또한 프로그람서고로 실현되여 있다. 이것은 임의의 대상에 대해 감시기의 걸쇠들을 놓을수 있도록 한다. 특히 련결식목록과 같은데서 모든 련결식목록을 하나의 걸쇠로 쇠를 채우거나 매개 목록에 하나의 걸쇠를 가지거나 매개 목록의 매개 요소에 하나의 걸쇠를 가지려고 할수도 있다.

Hoare의 판본을 먼저 보고 그다음 구체적으로 론의한다.

신호를 가진 감시기

감시기는 하나 또는 그이상의 수속, 사용순차 및 국부자료로 이루어 지는 쏘프트웨어모듈이다. 감시기의 주요기능은 다음과 같다. 즉

1. 국부자료변수들은 감시기의 수속들만 호출할수 있으며 외부수속은 호출할수 없다.
2. 프로쎄스는 자기의 수속들중에서 하나를 기동시켜 감시기에 들어 간다.
3. 한번에 하나의 프로쎄스만을 감시기에서 집행할수 있는데 감시기를 기동시킨 임의의 다른 프로쎄스도 중단되여 감시기가 사용가능해 지기를 기다린다.

첫 두개의 기능은 객체지향쏘프트웨어에서 객체들의 기능을 련상시킨다. 사실 객체지향조작체계나 프로그람작성언어는 특수한 기능을 가진 객체로서 감시기를 쉽게 실현할수 있다.

감시기는 한번에 하나의 프로쎄스규률을 적용하여 호상배제기능을 보장할수 있다. 감시기에서 자료변수는 한번에 하나의 프로쎄스만 호출할수 있다. 그러므로 공유된 자료구조를 감시기안에 놓아 그것을 보호할수 있다. 만일 감시기에서 자료가 일정한 자원을 표현한다면 감시기는 자원에 접근하는데서 호상배제기능을 준다.

병행처리에서 편리하도록 하기 위해 감시기는 동기화도구를 가지고 있어야 한다. 실례로 프로쎄스가 감시기를 기동시키며 한편 감시기에서 일정한 조건이 만족될 때까지 중단되여야 한다고 가정하자. 프로쎄스가 중단될뿐아니라 일부 다른 프로쎄스들이 들어가도록 감시기를 해방하는 기능이 요구된다. 후에 조건이 만족되고 감시기가 다시 사용할수 있을 때 프로쎄스는 그의 중단점에서 계속하여 감시기에 재진입할것을 요구한다.

감시기는 조건변수들을 사용하여 동기화를 지원하는데 변수들은 감시기안에 있으며 감시기안에서만 호출할수 있다. 두가지 기능이 조건변수에 따라 동작한다. 즉

- *cwait (c)*: 조건 *c* 에서 호출중인 프로쎄스의 집행을 중단시킨다. 감시기는 이제부터 다른 프로쎄스가 사용할수 있다.
- *csignal(c)*: 같은 조건에서 *cwait*다음에 중단된 일부 프로쎄스의 집행을 회복한다. 그러한 프로쎄스가 몇개 있다면 그중에서 하나를 선택하고 그러한 프로쎄스가 전혀 없다면 아무것도 하지 않는다.

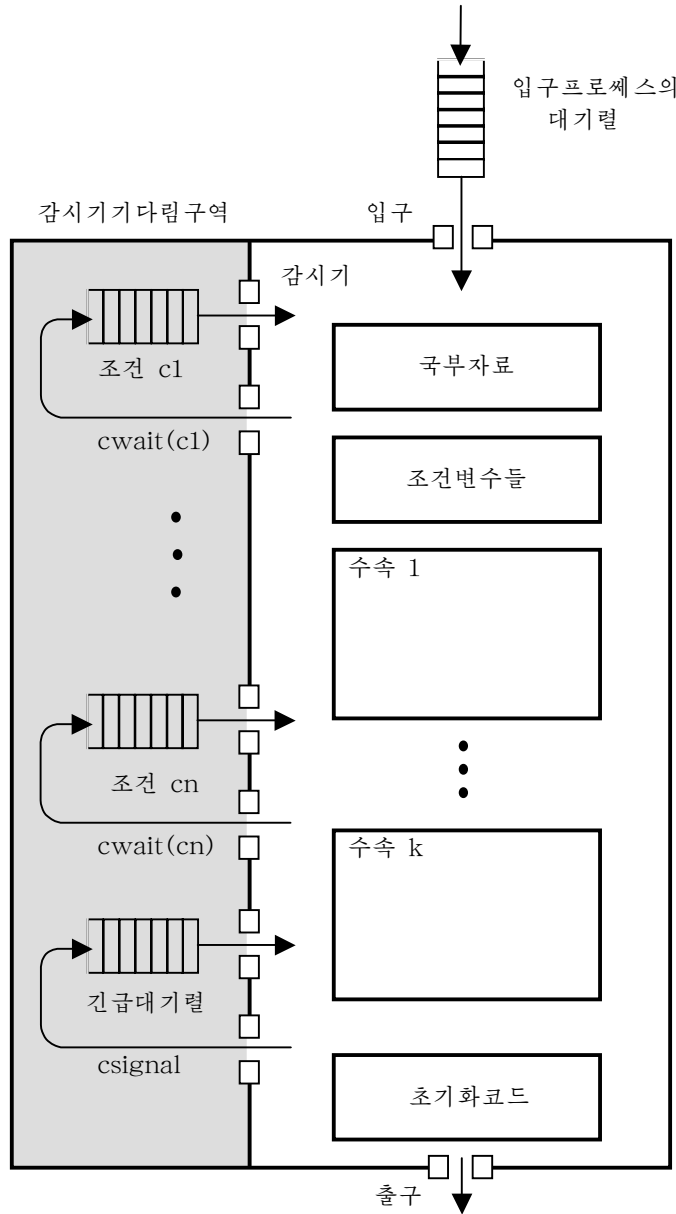



그림 5-21. 감시기의 구조

감시기의 *wait/signal* 조작은 신호기의 조작과 다르다. 만일 감시기에서 프로쎄스가 신호하고 조건변수를 기다리는 과제가 전혀 없으면 신호는 손실된다.

그림 5-21은 감시기의 구조를 보여 준다. 프로쎄스가 자기의 수속들중에서 임의의것을 기동시켜 감시기에 들어 갈수 있다고 하여도 감시기가 한번에 하나의 프로쎄스만 감시기에 있게 감시를 받는 단일한 입구점을 가지고 있다고 생각할수 있다. 감시기에 들어가려는 다른 프로쎄스들은 감시기의 사용가능성을 기다리면서 중단된 프로쎄스들의 대기렬에 넣는다. 일단 프로쎄스가 감시기에 있으면 그것은 *cwait*(*x*)를 내보내여 조건 *x*에

서 그자체를 일시적으로 중단시킬수 있는데 그것은 그다음에 조건이 변할 때 감시기에 다시 들어 가려고 기다리는 프로쎄스의 대기렬에 놓인다.

만일 감시기에서 집행하고 있는 프로쎄스가 조건변수 *x*에서 어떤 변화를 검출하면 *csignal*(*x*)를 내보내는데 이것은 대응하는 조건을 가진 대기렬에 조건이 변화되였다는 것을 통지한다.

감시기를 사용하는 실례로 유한한 완충기의 생산자/소비자문제에로 화제를 돌리자. 그림 5-22는 감시기를 사용하는 풀이를 보여 주고 있다. 감시기모듈인 경계완충기는 문자를 기억시키는데 또는 되찾는데 사용되는 완충기를 조종한다. 감시기는 두개의 조건변수를 가지고 있다. 즉 *notfull*은 완충기에 적어도 하나의 문자를 추가할수 있는 공간이 있을 때 참으로 되며 *notempty*는 완충기에 적어도 하나의 문자가 있을 때 참으로 된다.

생산자는 감시기안에 있는 수속 *append*에 의해서만 문자를 완충기에 추가할수 있으며 직접 접근하지 않는다. 수속은 우선 조건 *notfull*을 검사하여 완충기에 사용할수 있는 공간이 있는가를 판정한다. 만일 없으면 감시기를 집행하는 프로쎄스는 그 조건에서 중단된다. 일부 다른 프로쎄스(생산자 또는 소비자)는 이제부터 감시기에 들어 갈수 있다. 후에 완충기가 더이상 차지 않을 때 중단된 프로쎄스를 대기렬에서 꺼내여 다시 활성화시키고 처리를 계속해 나갈수 있다. 문자를 완충기에 넣은 다음 프로쎄스는 *notempty*조건을 신호한다. 류사한 설명을 소비자의 기능에 대해서도 할수 있다.

이 실례는 감시기를 사용하여 동작을 분할하는것을 신호기와 비교하여 보여 준다. 감시기의 경우에 감시기의 구성자체가 호상배제를 수행한다. 즉 생산자와 소비자가 둘다 동시에 완충기에 접근하는것은 불가능하다. 그러나 프로그람작성자는 적절한 *cwait*와 *csignal*기본지령을 감시기안에 놓아 프로쎄스가 꽉 찬 완충기에 항목을 예약하거나 빈 완충기에서 항목을 꺼내지 못하도록 보호해야 한다. 신호기인 경우에 호상배제와 동기화는 둘다 프로그람작성자가 한다.

그림 5-22에서 프로쎄스가 *csignal*기능을 집행한후에 즉시 감시기를 벗어 난다는데 주목하자. 만일 *csignal*이 수속의 끝에서 발생하지 않으면 Hoare의 제안에서 신호를 내보내는 프로쎄스는 중단되여 감시기를 사용할수 있도록 하며 감시기가 자유로울 때까지 대기렬에 놓인다. 이 점에서 하나의 가능성은 중단된 프로쎄스를 입구대기렬에 놓아 그것이 아직 감시기에 들어 가지 못한 다른 프로쎄스들과의 접근을 완료하도록 하는데 있을것이다. 그러나 *csignal*기능에 따라 중단된 프로쎄스는 이미 감시기에서 자기의 과제를 부분적으로 수행하였으므로 개별적인 긴급대기렬을 설치하여 프로쎄스가 새로 들어가는 프로쎄스들을 선행하도록 한다(그림 5-21). 감시기를 사용하는 하나의 언어로서 Concurrent Pascal은 csignal이 다만 감시기수속이 집행한 마지막조작으로 나타날것을 요구한다.

조건 x를 기다리고 있는 프로쎄스가 전혀 없으면 *csignal*(x)에 대한 집행은 아무런 효과도 내지 못한다.

신호기를 사용할 때처럼 감시기의 동기화기능에서 오유가 생길수 있다. 실례로 경계완충기감시기에서 어느 하나의 *csignal*기능이 빠진다면 대응하는 조건대기렬에 들어가는 프로쎄스는 영원히 요청접수가 중단된다. 신호기에 비한 감시기의 우점은 모든 동기화기능이 감시기에 국한된다는것이다. 따라서 동기화가 정확히 집행되였는가를 확증하기 쉬우며 오유를 검출하기 쉽다. 게다가 일단 감시기가 정확히 프로그람화되면 보호자원에 대한 접근은 모든 프로쎄스로부터의 접근에 대해 정확하다.

```
/*생산자소비자프로그람*/
monitor boundedbuffer;
char buffer [N];                               /*N개항목을 위한 공간*/
int nextin, nextout;                       /*완충기의 지시기*/
int count;                                 /*완충기안에서 항목의 수*/
int notfull, notempty;                     /*동기화용*/

void append(char x)
{
if(count= =N)
        cwait(notfull);          /*완충기가 꽉 찼다. 웃한계초과를 피하시오.*/
buffer[nextin]=x;
nextin=(nextin+1)%N;
count++;                                 /*완충기안에 하나이상의 항목*/
csignal(notempty);                         /*임의의 기다리는 소비자를 회복하시오.*/
}
void take(char x)
{
if (count= =0)
        cwait(notempty);                   /*완충기가 비여 있다. 아래한계초과를 피하시오.*/
x=buffer=[nextout];
nextout=(nextout+1)%N;
count--;                                   /*완충기에 수개의 항목*/
csignal(notfull);                /*임의의 기다리는 수속을 회복하시오.*/
}
{                                          /*감시기의 본체*/
nextin=0; nextout=0; count=0;               /*완충기는 초기에 비여 있다.*/
}
void producer()
char x;
{
while(true)
        {
        procduce(x);
        append(x);
}
}
void consumer()
{
char x;
while(true)
{
        take(x);
        consume(x);
}
}
void main()
{
parbegin(producer, consumer);
}
```

통지와 방송을 가진 감시기

감시기에 대한 Hoare의 정의[HOAR 74]는 적어도 하나의 프로쎄스가 조건대기렬에 있다면 대기렬에서 나오는 프로쎄스는 또 다른 프로쎄스가 그 조건에서의 *csignal*을 내보낼 때 즉시에 실행할것을 요구한다. 결국 *csignal*을 내보내는 프로쎄스는 즉시 감시기를 벗어 나든가 감시기상에서 중단되여야 한다.

이 방법에는 두가지 약점이 있다. 즉

1. *csignal*을 내보내는 프로쎄스가 감시기에서 끝나지 않았다면 두개의 추가적인 프로쎄스절환을 요구한다. 즉 하나는 프로쎄스를 중단시키는것이고 다른 하나는 감시기를 사용할수 있을 때 그것을 재개하는것이다.
2. 신호와 관련된 프로쎄스의 일정작성은 완전히 믿음성 있어야 한다. *csingnal*을 내보낼 때 대응하는 조건대기렬에서 나오는 프로쎄스는 즉시에 활성화되여야 하며 일정작성에는 그 어떤 다른 프로쎄스도 활성화되기전에 감시기에 들어 가지 않는다는것을 담보해야 한다. 그렇지 않으면 프로쎄스가 활성화되는 조건이 변할수 있다. 실례로 그림 5-22에서 *csignal*(*notempty*)을 내보낼 때 *notempty* 대기렬에서 나오는 프로쎄스는 새로운 소비자가 감시기에 들어 가기전에 활성화되여야 한다. 또 다른 실례를 보자. 생산자프로쎄스는 어떤 문자를 추가하고 그 다음 신호하기전에 실패할수 있는데 *notempty*대기렬에서 어떤 프로쎄스는 영원히 요청접수중단 당하게 된다.

Lampson 및 Redell은 Mesa언어에서의 서로 다른 감시기에 대한 정의를 내놓았다[LAMP 80]. 그들이 내놓은 방법은 방금 설명한 문제를 극복하며 몇가지로 쓸모 있게 확장할수 있다. Mesa언어의 감시기구조는 또한 Modula-3체계의 프로그람작성언어에서 쓰인다[NELS 91]. Mesa언어에서 *csignal*기본지령은 *cnotify*로 교체되는데 그것은 다음과 같이 해석할수 있다. 즉 감시기에서 집행중인 프로쎄스가 *cnotify*(*x*)를 집행할 때 *x*의 조건대기렬은 통지를 받지만 신호하는 프로쎄스는 계속 집행한다. 통지의 결과는 조건대기렬의 머리부에 있는 프로쎄스가 감시기를 사용할수 있는 어떤 편리한 앞으로의 시기에 재개될것이라는 바로 그것이다. 그러나 어떤 다른 프로쎄스가 기다리는 프로쎄스보다 먼저 감시기에 들어 가지 않으리라는 그 어떤 담보도 없기때문에 기다리는 프로쎄스는 조건을 다시 검사해야 한다. 실례로 경계완충기감시기에서 수속은 이제부터 그림 5-23의 코드를 가지게 된다.

if문은 **while**문으로 교체된다. 결국 이 명령은 조건변수에 대한 적어도 한번의 과잉평가를 가져 온다. 그러나 이번에는 아무런 과잉프로쎄스절환도 없으며 기다리고 있는 프로쎄스가 *cnotify*후에 실행해야 한다는 아무런 제한도 없다.

*cnotify*기본지령과 련관시킬수 있는 하나의 유용한 수단은 매개 조건기본지령과 련관된 감시용시계이다. 최대시간초과간격을 기다리고 있었던 프로쎄스는 조건을 통지 받았든 안받았든 관계없이 준비상태에 놓인다. 프로쎄스는 활성화될 때 조건을 검사하며 조건이 만족되면 계속 집행한다. 시간초과는 다른 프로쎄스가 조건을 신호하기전에 실패하는 사건에서 프로쎄스의 애매한 고갈을 막아 준다.

프로쎄스가 강제로 재활성화되는것이 아니라 통지를 받게 되는 규칙을 사용하여 *cb roadcast*기본지령을 집결지에 추가할수 있다. 방송은 조건을 기다리는 모든 프로쎄스가 준비상태에 놓이도록 한다. 이것은 프로쎄스가 얼마나 많은 다른 프로쎄스들이 재활성화되겠는지 모르는 정황에서 편리하다. 실례로 생산자/소비자프로그람에서 추가 및 꺼내기 기능이 둘다 가변길이를 가진 문자블로크에 적용할수 있다고 가정하자. 그 경우에 만

```
void append(char x)
  {
  while(count == N)
          cwait(notfull);          /*완충기가 꽉 찼다. 웃한계초과를 피하시오.*/
  buffer[nextin]=x;
  nextin=(nextin+1)%N;
  count++;                              /*완충기안에 하나이상의 항목*/
  csignal(notempty);                        /*임의의 기다리는 소비자를 회복하시오.*/
  }

  void take(char x)
  {
  while (count= =0)
          cwait(notempty);                  /*완충기가 비여 있다. 아래한계초과를 피하시오.*/
  x=buffer=[nextout];
  nextout=(nextout+1)%N;
  count--;                                  /*완충기에 수개의 항목*/
  csignal(notfull);                /*임의의 기다리는 수속에 통지하시오.*/
  }
```

그림 5-23. 경계완충기의 감시기

일 생산자가 문자블로크를 완충기에 추가한다면 기다리고 있는 매 소비자가 얼마나 많은 문자를 소비하기 위해 준비되여 있는가를 알 필요가 없다. 그것은 단순히 *cbroadcast*를 내보내고 모든 기다리고 있는 프로쎄스들에 다시 시도하도록 경고한다.

게다가 어느 다른 프로쎄스가 재능동화되는가를 프로쎄스가 정확히 도출해 내기 힘들 때 방송을 사용할수 있다. 한가지 좋은 실례로 기억기관리자를 들수 있다. 관리자는 *j*개의 자유바이트를 가지는데 프로쎄스가 보충적인 *k*개의 바이트를 자유롭게 해주지만 어느 기다리는 프로쎄스가 총 *k+j*개의 바이트를 가지고 진행할수 있는지 알지 못한다.

호어감시기에 대한 램슨/리델감시기의 우점은 램슨/리델의 방법이 오유를 더 적게 낸다는것이다. 램슨/리델방법에서는 매개 수속의 신호를 받고 후에 감시기의 변수를 검사하며 **while**구조를 사용하여 프로쎄스가 신호방식프로그람에서 오유를 발생시키지 않고 부정확하게 신호하거나 방송할수 있다. 신호방식프로그람은 의의 있는 변수를 검사할것이며 만일 요구하는 조건이 만족되지 않으면 계속 기다리게 될것이다.

램슨/리델감시기의 또 다른 우점은 그것이 프로그람구성에서보다 모듈적인 방법에로 지향하고 있다는것이다. 실례로 완충기배정기를 고찰하여 보자. 순차적인 프로쎄스들을 협동시키는데서 만족되여야 할 두개의 조건준위가 있다.

1. 모순이 없는 자료구조. 이로 인하여 감시기는 호상배제를 수행하며 완충기에 관하여 다른 조작을 허용하기전에 입력 또는 출력조작을 완료한다.
2. 1준위와 그밖에 프로쎄스가 그의 배정요청을 완료하는데 충분한 기억기

호어의 감시기에서 매개 신호는 1준위조건을 나르면서 또한 《나는 당신의 특정한 배정호출이 이제부터 작업하는데 충분한 바이트들을 해방하였다.》는 암시적통보문을 나른다. 따라서 신호는 2준위의 조건을 암시적으로 나른다. 만일 프로그람작성자가 후에 2준위의 조건에 대한 정의를 변화시킨다면 모든 신호방식프로쎄스들은 프로그람을 다시 작성해야 할것이다. 만일 프로그람작성자가 어떤 특정한 기다리고 있는 프로쎄스가 만든 가정을 변화시킨다면(즉 약간 차이나는 2준위가 불변이기를 기다리고 있는) 모든 신호방

식프로쎄스들을 다시 프로그람작성해야 할수도 있다. 이것은 비모듈적이며 코드가 수정될 때 동기화오유(실례로 착오에 의한 기동)를 일으킬수도 있다. 프로그람작성자는 2준위조건에서 작은 변화가 생길 때마다 기억하고 모든 프로쎄스를 수정해야 한다. 이제부터는 램슨/리델의 감시기를 사용하여 방송이 1준위의 조건을 담보해 주며 2준위가 유지할수 있다는 암시를 나르는데 매개 프로쎄스는 2준위의 조건 그자체를 검사한다. 만일 기다리는자든지 신호발송자든지에서 어떤 변화가 2준위조건으로부터 생긴다면 매개 수속이 그자체의 2준위조건을 검사하므로 그 어떤 잘못된 기동이 일어 날수 없다. 따라서 2준위의 조건을 매개 수속안에서 은페시킬수 있다. 호어의 감시기를 사용하여 2준위조건을 기다리는 프로쎄스로부터 매개 신호방식코드에 날라야 하는데 이것은 자료추출과 수속사이의 모듈성원칙들을 위반한다.

제 6 절. 통보문넘기기

프로쎄스들이 서로 대화할 때 두가지 기본요구 즉 동기화와 통신요구를 만족시켜야 한다. 호상배제를 수행하기 위해 프로쎄스들을 동기화시켜야 하는데 협동프로쎄스들이 요구하여 정보를 변화시킬수도 있다. 이 기능을 둘다 보장하는 한가지 방법이 통보문넘기기이다. 통보문넘기기는 분산체계에서는 물론 공유기억기식다중처리기 및 단일처리기체계에서의 실현에 그자체를 보태준다는 우점을 더 가지고 있다.

통보문넘기기체계에는 많은 형식이 있다. 이 절에서는 일반적인 소개를 하는데 이것은 그러한 체계들에서 대표적으로 보게 되는 기능들을 설명해 준다. 통보문넘기기의 실제적기능은 보통 한쌍의 기본지령형식으로 주어 진다. 즉

send（destination，message）
receive（source，message）

이것은 프로쎄스가 통보문넘기기에서 예약하는 최소한의 조작모임이다. 프로쎄스는 통보문의 형식으로 목적지로 지정된 다른 프로쎄스에 정보를 보낸다. 프로쎄스는 보내는 프로쎄스의 원천과 통보문을 가리키는 수신기본지령을 집행하여 정보를 수신한다.

통보문넘기기체계와 관련이 있는 몇가지 설계물이 표 5-4에 제시되여 있는데 이 절의 나머지 부분에서는 이 매개 설계물들을 차례로 론의한다.

동기화

두 프로쎄스사이에서 통보문의 통신은 둘사이에서 어떤 준위의 동기화를 암시해 준다. 즉 수신기는 다른 프로쎄스가 송신하기전에는 통보문을 수신할수 없다. 게다가 송신 또는 수신기본지령을 내보낸후에 프로쎄스에 무슨 일이 일어 나는지 지정해야 한다.

송신기본지령을 먼저 고찰해 보자. 송신기본지령이 프로쎄스에서 집행될 때 두가지가능성이 있다. 즉 송신하는 프로쎄스가 통보문이 수신될 때까지 페색되든가 또는 그렇지 않은것이다.류사하게 프로쎄스가 수신기본지령을 내보낼 때 두가지 가능성이 있다. 즉

1. 통보문이 이미 송신되였다면 그 통보문은 수신되며 집행이 계속된다.
2. 기다리는 통보문이 전혀 없다면 (ㄱ) 통보문이 도착할 때까지 프로쎄스가 페색되거나 (ㄴ) 프로쎄스가 계속 집행하며 수신하려는 시도를 버린다.

표 5-4. 프로쎄스사이의 통신 및 동기화를 위한 통보문체계의 설계특성

동기화	**서식**
송신	내용
페색식	길이
비페색식	고정
수신	가변
페색식	
비페색식	**기다림규률**
도착에 대한 시험	FIFO
주소지정	우선권
직접	
송신	
수신	
로출식	
암시식	
간접	
정적	
동적	
소유	

그러므로 송신자와 수신자는 둘다 페색 또는 비페색중일수 있다. 어떤 특정한 체계가 보통 하나 또는 두개의 결합으로 실현되는 경우가 있지만 세개의 결합이 일반적이다. 즉

- **송신페색,수신페색 :** 송신자와 수신자는 둘다 통보문이 배달될 때까지 페색되는데 이것을 때때로 상봉이라고 한다. 이 결합은 프로쎄스사이에서 엄격한 동기화를 할수 있게 한다.
- **송신비페색,수신페색 :** 송신자가 계속할수 있다고 하여도 수신자는 요청되는 통보가 도착할 때까지 페색된다. 이것이 아마도 가장 유용한 결합으로 된다. 그것은 프로쎄스가 하나 또는 그이상의 통보문을 여러가지 목적지에 가능한 빨리 송신할수 있게 해준다. 유용한 작업을 하기전에 통보문을 수신해야 하는 프로쎄스는 그러한 통보문이 도착할 때까지 페색되여야 한다. 실례로 다른 프로쎄스들에 봉사 또는 자원을 보장해 주기 위한 봉사기프로쎄스를 들수 있다.
- **송신비페색,수신비페색 :** 어느쪽도 기다리려고 하지 않는다.

송신비페색은 많은 병행프로그람식 과제들에서 가장 자연스럽다. 실례로 만일 그것을 인쇄와 같은 출력조작을 요청하는데 사용한다면 요청하는 프로쎄스가 통보문의 형식으로 그 요청을 내보내고 그다음 계속할수 있게 해준다. 송신비페색의 한가지 위험은 어떤 오유는 프로쎄스가 통보문을 반복하여 산생시키는 상태를 가져 올수 있다는것이다. 프로쎄스에 규률을 적용하는 그 어떤 페색방식도 없으므로 통보문들은 처리기시간과 완충기의 공간을 포함하여 체계의 자원을 다른 프로쎄스와 조작체계에 손해가 되게 소비할수 있다. 또한 송신비페색은 프로그람작성자에게 부담을 주어 통보문이 수신되였다는것을 결정한다. 즉 프로쎄스가 응답통보문을 사용하여 통보문의 접수를 알려 주어야 한다.

수신기본지령에서 페색식판본이 많은 병행프로그람식과제들에서 가장 보편적이다. 일반적으로 통보문을 요청하는 프로쎄스는 진행하기전에 기대되는 정보를 요구한다. 그러나 만일 통보문이 상실된다면(이런 현상은 분산된 체계에서 일어 날수 있다.) 또는 프

로쎄스가 예상되는 통보문을 보내기전에 실패한다면 수신하는 프로쎄스가 불명확하게 페색될수 있다.수신비페색을 사용하여 이 문제를 해결할수 있다. 그러나 이 방법의 위험성은 만일 프로쎄스가 이미 정합수신을 진행한후에 통보문을 송신하면 그 통보문이 상실된다는것이다. 다른 가능한 방법은 수신을 하기전에 통보문이 기다리고 있는가 아닌가를 프로쎄스가 검사하며 프로쎄스가 수신기본지령에서 하나이상의 자원을 지정하도록 하는것이다. 프로쎄스가 하나이상의 자원으로부터 통보문을 기다리고 있다면 후자의 방법이 유용하며 통보문중의 어떤것이 도착하면 진행할수 있다.

주소지정

명백히 프로쎄스가 통보문을 수신하는 송신기본지령문에서 어떤 지정방법이 있어야한다. 류사하게 대부분의 실현은 수신하는 프로쎄스가 수신되는 통보문의 자원을 가리키도록 한다.

송신 및 수신기본지령에서 프로쎄스를 지정하는 여러가지 방안들은 두개의 범주 즉 직접주소지정방식과 간접주소지정방식에 속한다. **직접주소지정방식**에 대해서 말한다면 송신기본지령이 목적지프로쎄스에 대한 특정한 식별자를 가진다. 수신기본지령은 두 방법중의 하나로 처리될수 있다. 하나의 가능성은 프로쎄스가 송신하는 프로쎄스를 로출된 방법으로 지명하도록 요구하는것이다. 그리하여 프로쎄스가 어느 프로쎄스로부터 통보문을 기대하고 있는가를 사전에 확실하게 알수 있다. 이것은 흔히 협동하는 병행프로쎄스에서 효과적일것이다. 그러나 다른 경우에 예상되는 원천프로쎄스를 지정할수 없다. 실례로 인쇄기-봉사기프로쎄스를 들수 있는데 이것은 다른 프로쎄스로부터 인쇄요청통보문을 접수한다. 그러한 응용프로그람에서 보다 효과적인 방법은 암시적 주소지정방식을 사용하는것이다. 이 경우에 수신기본지령의 원천파라메터는 수신조작이 수행되였을 때 반환된 값을 가진다.

다른 일반적인 방법은 **간접주소지정방식**이다. 이 경우에 통보문은 송신자로부터 수신자에게 직접 송신되지 않고 일시적으로 통보문을 보관할수 있는 대기렬로 되여 있는 공유자료구조에로 송신된다. 그러한 대기렬을 일반적으로 우편통이라고 한다. 그리하여 두 프로쎄스가 통신할 때 하나의 프로쎄스는 통보문을 해당한 우편통에로 송신하며 다른 프로쎄스는 우편통에서 그 통보문을 꺼낸다.

간접주소지정방식을 사용하는것이 가지는 유리성은 송신기와 수신기를 분리시킴으로써 그것이 통보문사용에서 더 큰 유연성을 주도록 한다는것이다. 송신기와 수신기들사이의 관계는 1대 1, 다대 1, 1대 다 또는 다대 다관계로 될수 있다. 1대 1관계는 전용통신회선을 두 프로쎄스사이에 설치한다. 이것은 오유를 일으키는 다른 프로쎄스의 간섭으로부터 그것들의 호상작용을 격리시킨다. 다대 1관계는 의뢰기/봉사기대화에서 유용한데 하나의 프로쎄스가 수많은 다른 프로쎄스들에 봉사한다. 이 경우에 우편통을 흔히 포구라고 한다(그림 5-24). 1대 다관계는 하나의 송신기와 여러개의 수신기를 허용하는데 그것은 통보문이나 정보를 프로쎄스의 모임에 방송하는 응용프로그람들에서 쓸모 있다.

우편통에 대한 프로쎄스들의 련계는 정적일수도 있고 동적일수도 있다. 포구는 흔히 특정한 프로쎄스와 정적으로 련결된다. 즉 포구는 창조되여 프로쎄스에 영원히 할당된다. 류사하게 1대 1관계는 대체로 정적 및 영구적으로 정의된다. 송신기들이 많을 때 우편통에 대한 송신기의 련계는 동적으로 발생할수 있다. *connect* 및 *disconnect*와 같은 기본지령을 이 목적에 사용할수 있다.

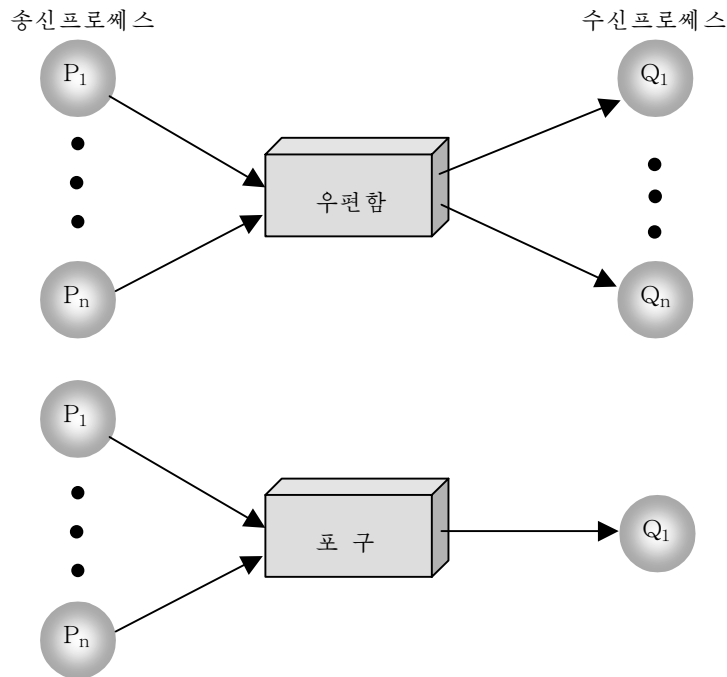



그림 5-24. 간접프로쎄스의 통신

련관된 발행은 우편통에 대한 소유권을 가지고 하여야 한다. 포구인 경우에는 대체로 프로쎄스가 소유하고 프로쎄스에 의해 창조된다. 이런데로부터 프로쎄스가 파괴되면 포구도 파괴된다. 일반우편통인 경우에 조작체계는 우편통창조봉사를 할수 있다. 그러한 우편통들은 창조하는 프로쎄스가 소유하고 있는것으로 볼수 있는데 이 경우에 그것들은 프로쎄스와 같이 움직인다. 또는 그러한 우편통들을 조작체계가 소유하고 있는것으로 볼수 있는데 이 경우에 그 우편통을 파괴하는데는 어떤 공개적인 지령이 있어야 할것이다.

통보문의 형식

통보문의 형식은 통지하는 기능의 목적과 그 기능이 단일콤퓨터상에서 실행하는가 또는 분산체계상에서 실행하는가 하는데 관계된다. 설계자들은 일부 조작체계들에서 처리 및 기억기의 간접소비시간을 최소화하기 위해 짧고 고정길이를 가진 통보문을 제기하였다.만일 많은 자료를 넘기려고 한다면 그 자료를 하나의 파일로 놓을수 있으며 통보문은 그 파일을 단순히 참조한다. 보다 유연한 방법은 가변길이의 통보문을 사용하는것이다.

그림 5-25는 가변길이의 통보문을 지원하는 조작체계에서의 대표적인 통보문형식을 보여 주고 있다. 이 통보문은 두개의 부분 즉 통보문에 대한 정보를 담고 있는 머리부와 통보문의 실제적 내용을 담고 있는 본체로 나누어져 있다. 머리부는 통보문의 원천과 의도하는 목적지길이마당 및 여러가지 형태의 통보문을 구별하기 위한 형마당을 가질수 있다. 통보문의 련결목록을 창조할수 있는 지시기마당, 원천 및 목적지사이에서 넘어 간 통보문의 수와 차수에 대한 추적을 보존하기 위한 순차의 수 및 우선권마당과 같은 추가적인 조종정보도 있을수 있다.

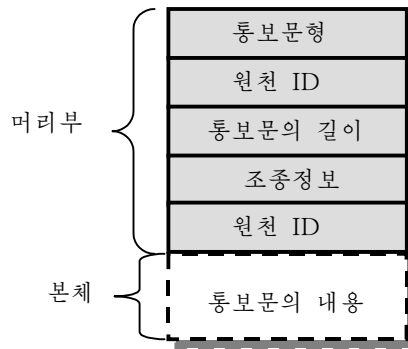



그림 5-25. 일반적인 통보문형식

기다림규률

가장 단순한 기다림규률은 선입선출이지만 어떤 통보문이 다른것보다 긴급하다면 이것이 충분하지 못할수 있다. 대리방안은 통보문에 기초하거나 송신기가 지명하여 통보문의 우선권을 지정하도록 하는것이다. 다른 하나의 대리방안은 수신기가 통보문대기렬을 조사하고 다음에 수신할 통보문을 선택하도록 하는것이다.

호상배제

그림 5-26은 호상배제를 실시하는데 통보문넘기기를 사용할수 있는 한가지 방법을 보여 주고 있다(그림 5-1, 5-5 및 5-9와 비교하시오.). 페색식수신기본지령과 비페색식송신기본지령을 사용하는것으로 한다. 병행프로쎄스의 모임은 우편통인 mutex를 공유하는데 이것은 송신 및 수신하는데서 모든 프로쎄스가 사용할수 있다. 우편통은 초기화되여 빈 내용으로 되여 있는 단일통보문을 포함하고 있다. 자기의 림계구간에 들어 가려고 하는 프로쎄스는 우선 어떤 통보문을 수신하려고 한다. 만일 우편통이 비여 있으면 프로쎄스는 페색된다. 일단 프로쎄스가 통보문을 얻었으면 자기의 림계구간을 수행하고 통보문을 우편통에 되돌려 놓는다. 그리하여 통보문은 프로쎄스에서 프로쎄스에로 통과하는 통표와 같은 기능을 수행한다.

```
/*호상배제프로그람*/
const int n = /*프로쎄스의 수*/;
void P(int I)
{
    message msg;
    while (true)
    {
        receive(mutex, msg);
        /*림계구간*/
        send(mutex, msg);
        /*나머지*/
    }
}
void main()
```

```
{
    create_mailbox(mutex);
    send(mutex, null);
    parbegin(P(1),P(2),...,P(n));
}
```

그림 5-26. 통보문을 사용하는 호상배제

앞의 풀이는 만일 하나이상의 프로쎄스가 수신조작을 병행하여 수행한다면 그때

- 통보문이 있으면 그것이 단지 하나의 프로쎄스에 분배되고 다른것들은 페색되거나
- 통보문대기렬이 비여 있으면 모든 프로쎄스가 페색되는데 통보문을 사용할수 있을 때 하나의 페색된 프로쎄스만이 활성화되여 통보문을 받는다.

```
Const int
    Capacity=/*완충용량*/
    Null=/*빈통보문*/
Int I;
Void producer()
{   message pmsg;
    while(true)
    {
        receive(mayproduce, pmsg);
        pmsg=produce();
        send(mayconsume, pmsg);
    }
}
void consumer()
{   message cmsg
    while(true)
    {
        receive(mayconsume,cmsg);
        consume(cmsg);
        send(mayproduce,null);
    }
}

void main()
{
    create_mailbox(mayproduce);
    create_mailbox(mayconsume);
    for(int I=1;I<=capacity; I++)
      send(mayproduce,null);
    parbegin(producer,consumer);
}
```

그림 5-27. 통보문을 사용하는 경계완충기식의 생산자/소비자문제에 대한 풀이

이 가정은 실제적으로 모든 통보문넘기기기능에 맞는다.

통보문넘기기를 사용하는 또 다른 실례로서 그림 5-27이 경계완충기식의 생산자/소비자 문제에 대한 풀이로 된다. 통보문넘기기의 기초적인 호상배제기능을 사용하여 그림 5-16과 류사한 알고리듬을 가지고 문제를 해결할수 있었다. 그대신에 그림 5-27의 프로그람은 통보문넘기기를 신호외에 자료를 넘기는데 사용할수 있다는 우점이 있다. 두개의 우편통을 사용한다. 생산자가 자료를 발생시키는데 따라 그것을 우편통 *mayconsume*에 통보문으로 보낸다. 적어도 하나의 통보문이 우편통에 있는동안 사용자가 소비할수 있다. 이로부터 *mayconsume*은 완충기로 봉사하는데 완충기안의 자료는 통보문렬로 구성된다. 완충기의 《크기》는 전역변수 *capacity*가 확정한다. 초기에 우편통 *mayproduce*는 완충기의 용량과 같은 수의 빈 통보문으로 차 있다. *mayproduce*에 있는 통보문의 수는 매개 생산에 따라 줄어 들며 매개 소비에 따라 확장된다.

이 방법은 아주 유연하다. 모두가 두 우편통에 접근하는한 여러개의 생산자/소비자가 있을수 있다. 지어 체계를 한 싸이트에는 모든 생산자프로쎄스와 *mayproduce*우편통으로 그리고 다른 싸이트에는 모든 소비자프로쎄스와 *mayconsume*우편통으로 분산시킬수도 있다.

제 7 절. 읽기자/쓰기자문제

동기화 및 병행성기구에 대한 설계를 취급하는데서 쓸모가 있는것은 알려 진 문제들에 가까운 문제를 관련시킬수 있고 이 알려 진 문제들을 해결하기 위한 능력으로서 어떤 풀이를 검사할수 있는 능력이다. 문헌에서 몇가지 문제들이 중요성을 가정하였고 자주 나타나는데 그것은 량자가 공통적인 설계문제들에 대한 실례로 되여 있기때문이며 그의 교육적가치때문이다. 그러한 한가지 문제가 생산자/소비자 문제인데 이것은 이미 론의되였다. 이 절에서는 또다른 고전적인 문제로서 읽기자/쓰기자문제를 본다.

읽기자/쓰기자문제는 다음과 같이 정의한다. 즉 몇개의 프로쎄스사이에서 공유된 자료구역이 있다. 자료구역은 파일, 주기억블로크 또는 지어 처리기등록기들의 기억단일수 있다. 자료구역을 읽어 내기만 하는 프로쎄스(읽기자프로쎄스)와 자료구역에 써넣기만 하는 프로쎄스(쓰기자프로쎄스)가 있다. 만족시켜야 할 조건은 다음과 같다. 즉

1. 임의의 개수의 읽기자프로쎄스들이 파일을 동시에 읽을수 있다.
2. 한번에 하나의 쓰기자프로쎄스만이 파일에 써넣을수 있다.
3. 쓰기자프로쎄스가 파일에 써넣고 있으면 그 어느 읽기자프로쎄스도 그것을 읽을 수 없다.

먼저 이 문제를 다른 두가지 문제 즉 일반적인 호상배제문제 및 생산자/소비자문제와 구별해 보자. 읽기자/쓰기자문제에서 읽기자프로쎄스는 자료구역에 써넣을수도 없고 쓰기자프로쎄스가 자료구역을 읽을수도 없다. 이런 경우를 포함하는 보다 일반적인 경우는 임의의 프로쎄스가 자료구역을 읽거나 써넣을수 있게 하는것이다. 그 경우에 림계구간으로 되는 자료구역에 접근하는 임의의 프로쎄스부분을 선언할수 있으며 일반적인 호상배제의 풀이가 있다. 보다 제한된 경우에 관심을 돌리는것은 이 경우에 보다 효과적인 풀이가 가능하기때문이다. 실례로 공유된 구역이 서고목록이라고 가정하자. 보통 서고사용자들은 목록을 읽고 책을 찾는다. 한명 또는 그이상의 사서들이 목록을 갱신할수 있다. 일반적인 풀이에서는 목록에 대한 매개의 접근을 림계구간으로 취급하며 사용자들은 한번에 하나의 목록을 읽어야 한다. 이것은 분명 무시할수 없는 지연을 가져 올것이다. 같은 시간에 쓰기자프로쎄스가 서로 간섭하지 못하게 하는것이 중요하며 써넣기가 진행중에 있는 동안 부정확한 정보에 대한 접근을 막기 위하여 읽기를 방지하는것이 또한 필요하다.

생산자/소비자문제를 단순히 단일쓰기자프로쎄스(생산자)와 단일읽기자프로쎄스(소비자)를 가진 읽기자/쓰기자문제의 특수경우로 고찰할수 있는가? 그 대답은《아니》이다. 생산자는 꼭 쓰기자프로쎄스가 아니다. 그것은 다음 항목을 어디에 써 넣는가를 판정하기 위해 대기렬의 지시기를 읽어야 하며 완충기가 다 차 있는가를 확정해야 한다. 류사하게 소비자는 꼭 읽기자프로쎄스가 아닌데 그것은 완충기로부터 하나의 단위를 꺼냈다는것을 알려 주기 위해 대기렬지시기를 조절해야 하기때문이다.

이제부터 이 문제에 대한 두가지 풀이를 론의하자

읽기자프로쎄스가 우선권을 가진다.

그림 5-28은 신호기를 사용하는 하나의 풀이로서 개개의 읽기자 및 쓰기자프로쎄스에 대한 하나의 실례를 보여 주는데 이 풀이는 여러개의 읽기자 및 쓰기자프로쎄스에서 변하지 않는다. 쓰기자프로쎄스는 단순하다. 신호기 *wsem*은 호상배제를 실시하는데 쓰인다. 하나의 쓰기자프로쎄스가 공유자료구역에 접근하고 있는 동안 그 어떤 다른 쓰기자프로쎄스 및 그 어떤 읽기자프로쎄스도 그것에 접근할수 없다. 읽기자프로쎄스는 또한 *wsem*을 사용하여 호상배제를 실시한다. 그러나 여러 읽기자프로쎄스를 허용하기 위해서는 그 어떤 읽기자프로쎄스도 읽기중이 아닐 때 읽으려고 하는 첫 읽기자프로쎄스가 *wsem*을 기다리도록 해야 한다. 이미 적어도 하나의 프로쎄스가 읽기중에 있을 때 다음의 읽기자프로쎄스는 들어 가기전에 기다릴 필요가 없다. 전역변수 *readcount*는 몇개의 읽기자프로쎄스에 대한 추적을 보존하는데 쓰이며 신호기 *x*는 *readcount*가 적절히 갱신되는것을 보증하는데 쓰인다.

```
/*읽기자 및 쓰기자프로그람 */
int readcount;
semaphore x=1, wsem=1;
void reader()
{
    while(true)
    {
        wait(x);
        readcount++;
        if(readcount= =1)
          wait(wsem);
        signal(x);
        READUNIT();
        Wait(x);
        readcount--;
        If(readcount= =0)
          signal(wsem);
        signal(x);
    }
 }
 void writer()
 {
    while(true)
```

```
        {
            wait(wsem);
            WRITEUNIT();
            signal(wsem);
        }
}
void main()
{
    readcount=0;
    parbegin(reader, writer);
 }
```

그림 5-28. 신호기를 사용하는 읽기자/쓰기자문제에 대한 풀이 :
읽기자프로쎄스가 우선권을 가진다.

쓰기자프로쎄스가 우선권을 가진다.

앞의 풀이에서는 읽기자프로쎄스가 우선권을 가진다. 일단 단일읽기자프로쎄스가 자료구역에 접근하기 시작했다면 적어도 하나의 읽기자프로쎄스가 읽기동작중에 있는 동안 읽기자프로쎄스들이 자료구역에 대한 조종을 유지할수 있다. 따라서 쓰기자프로쎄스들은 고갈을 면할수 없다.

그림 5-29는 일단 적어도 하나의 쓰기자프로쎄스가 써넣기를 선언했으면 그 어떤 새로운 읽기자프로쎄스에도 자료구역에 대한 접근을 허용하지 않는다는것을 담보하는 풀이를 보여 주고 있다. 쓰기자프로쎄스에서는 다음의 신호기와 변수들이 이미 정의된것에 추가된다. 즉

- 적어도 하나의 쓰기자프로쎄스가 자료구역에 대한 접근을 희망하고 있는 동안 모든 읽기자프로쎄스를 금지시키는 신호기 *rsem*
- *rsem*의 설정을 조종하는 변수 *writecount*
- *writecount*의 갱신을 조종하는 신호기 *y*

표 5-5. 그림 5-28의 프로그람에서 프로쎄스대기렬의 상태

읽기자프로쎄스만이 체계에 있다.	• *wsem*설정 • 아무러한 대기렬도 없다.
쓰기자프로쎄스만이 체계에 있다.	• *wsem*및 *rsem*설정 • 쓰기자프로쎄스들이 *wsem*에서 대기렬을 짓는다.
첫 읽기와 함께 읽기자 및 쓰기자프로쎄스가 둘다 있다.	• 읽기자프로쎄스가 *wsem*설정 • 쓰기자프로쎄스가 *rsem*설정 • 모든 쓰기자프로쎄스들이 *wsem*에서 대기렬을 짓는다. • 모든 쓰기자프로쎄스들이 *rsem*에서 대기렬을 짓는다. • 다른 읽기자프로쎄스들이*z*에서 대기렬을 짓는다.
첫 쓰기와 함께 읽기자 및 쓰기자프로쎄스가 둘다 있다.	• 쓰기자프로쎄스가 *wsem*설정 • 쓰기자프로쎄스가 *rsem*설정 • 쓰기자프로쎄스가 *wsem*에서 대기렬을 짓는다. • 하나의 읽기자프로쎄스가 *rsem*에서 대기렬을 짓는다. • 다른 읽기자프로쎄스들이 *z*에서 대기렬을 짓는다.

```
/*읽기자 및 쓰기자프로그람*/
int readcount, writecount;
semaphore x=1, y=1,z=1,wsem=1,rsem=1;
void reader()
{
   while(true)
   {
      wait(z);
      wait(rsem);
      wait(x);
      readcount++;
      if(readcount= =1)
      {
         wait(wsem);
      }
      signal(x);
      signal(rsem);
      signal(z);
      READUNIT();
      wait(x);
      readcount--;
      If(readcount= =0)
        signal(wsem);
      signal(x);
   }
 }
 void writer()
 {
    while(true)
    {
        wait(y);
        writecount++;
        if(writecount= =1)
           wait(rsem);
        signal(y);
        wait(wsem);
        WRITEUNIT();
        signal(wsem);
        wait(y);
        writecount--;
        if(writecount= =0)
          signal(rsem);
        signal(y);
     }
  }
  void main()
  {
      readcount=writecount=0;
      parbegin(reader, writer);
   }
```

그림 5-29. 신호기를 사용하는 읽기자/쓰기자문제에 대한 풀이 :
쓰기자프로쎄스가 우선권을 가진다.

읽기자에서는 하나의 추가적인 신호기가 요구된다. 긴 대기렬에는 *rsem*에 관하여 만들도록 허용하지 말아야 하는데 그렇지 않으면 쓰기자프로쎄스가 그 대기렬을 뛰여 넘을수 없을것이다. 따라서 하나의 읽기자프로쎄스만이 신호기 *z*에서 대기렬을 이루는 어떤 추가적인 읽기자프로쎄스들과 함께 *rsem*을 기다리기전에 즉시에 *rsem*에서 대기렬을 짓도록 허용한다. 표 5-5는 그 가능성들을 요약하고 있다.

```
void reader(int I)
{
  message rmsg;
      while(true )
      {
          rmsg=I;
          send(readrequest,rmsg);
          receive(mbox[I],rmsg);
          READUNIT();
          rmsg=I;
          send(finished,rmsg);
      }
}
void writer(int j)
{
  message rmsg;
      while(true)
      {
          rmsg=i;
          send(writerequest,rmsg);
          receive(mbox[j],rmsg);
          WRITEUNIT();
          rmsg=i;
          send(finished,rmsg);
      }
}
```

```
void controller()
{
    while(true)
    {
      if(count>0)
      {
        if(!empty(finished))
        {
          receive(finished,msg);
          count++;
        }
        else if(!empty(writerequest))
        {
          receive(writerequest,msg);
          writer_id=msg.id;
          cound=count-100;
        }
        else if(!empty(readrequest)
        {
            receive(readrequest,msg);
            count--;
            send(msg.id,"OK");
        }
       }
      if(count= =0)
      {
          send(writer_id,"OK");
          receive(finished,msg);
          count=100;
      }
      while(count<0)
      {
          receive(finished,msg);
          count++;
      }
    }
}
```

그림 5-30. 통보문넘기기를 사용하는 읽기자/쓰기자문제에 대한 풀이

쓰기자프로쎄스에 우선권을 주며 통보문넘기기를 사용하여 실현하는 대리풀이를 그림 5-30에서 보여 주고 있다. 이 경우에는 공유자료구역에 접근하는 조종기의 프로쎄스가 있다. 자료구역에 접근하려는 다른 프로쎄스들은 조종기에 요청통보문을 보내여 《OK》응답통보문으로 접근을 허용받으며 《완료》통보문으로 접근의 실현을 알려 준다. 조종기는 수신할수 있는 매개 통보문형에 하나씩 세개의 우편통으로 장비되여 있다.

조종기의 프로쎄스는 읽기요청통보문이 쓰기자프로쎄스에 우선권을 주기전에 쓰기요청통보문을 봉사한다. 그밖에 호상배제를 실시해야 한다. 이것을 하기 위하여 변수 *count*를 사용하는데 이것은 최대로 가능한 읽기자프로쎄스의 수보다 큰 어떤 수로 초기화된다. 이 실례에서는 100이라는 값을 사용한다. 조종기의 동작을 다음과 같이 요약할수 있다

- *count*>0이면 그 어떤 쓰기자프로쎄스도 기다리고 있지 않으며 능동적인 읽기자프로쎄스 가 있을수도 있고 없을수도 있다. 능동적인 읽기자프로쎄스를 명백히 하기위해 모든 《완료》통보문을 우선 봉사한다. 다음에 쓰기요청을 봉사하고 그 다음에 읽기요청을 봉사한다.
- *count*=0이면 미해결의 요청만이 쓰기요청이다. 쓰기자프로쎄스가 착수하여 《완료》통보문을 기다리도록 허용한다.
- *count*<0이면 쓰기자프로쎄스가 어떤 요청을 제기하였고 능동적인 모든 프로쎄스를 명백히 하기 위해 기다린다. 따라서 《완료》통보문만을 봉사하여야 한다.

요약, 기본용어 및 복습문제

현대 조작체계의 중심은 다중프로그람작성, 다중처리 및 분산처리이다. 여기에서 기본이며 조작체계설계수법에서 기본은 병행성이다. 여러 프로쎄스가 병행하여 집행중에 있을 때 실제적으로는 다중처리기체계든 가상적으로 단일처리기의 다중프로그람식체계든 충돌해결 및 협동문제가 생긴다.

병행프로쎄스는 몇가지 방법으로 대화할수 있다. 서로 알지 못하는 프로쎄스들은 처리기의 시간이나 입출력장치에 대한 접근과 같은 자원을 놓고 무관계하게 경쟁할수 있다. 프로쎄스들은 주기억기블로크나 파일과 같은 공통적인 객체들에 대한 접근을 공유하기 때문에 간접적으로 서로 알수 있다. 끝으로 프로쎄스들은 정보를 교환함으로써 직접적으로 서로 알거나 협동할수 있다. 이 대화들에서 생기는 기본문제점은 호상배제와 교착이다.

호상배제는 병행프로쎄스의 모임이 있으며 임의의 시각에 그중에서 하나만이 주어진 자원에 접근할수 있거나 주어 진 기능을 수행할수 있다는 하나의 조건이다. 호상배제수법을 자원에 대한 경쟁과 같은 충돌을 해결하는데 또는 프로쎄스들이 협동할수 있도록 동기화시키는데 사용할수 있다. 후자의 실례가 생산자/소비자모형인데 여기서는 하나의 프로쎄스가 자료를 완충기에 넣고 하나 또는 그이상의 프로쎄스가 완충기에서 자료를 꺼낸다.

호상배제를 보장하기 위한 몇가지 쏘프트웨어알고리듬이 개발되여 있는데 그중에서 가장 널리 알려 진것이 Dekker알고리듬이다. 쏘프트웨어적방법은 처리의 간접소비시간이 많을수 있으며 론리적오유(착오)가 많다. 호상배제를 지원하는 두번째 방법은 특수목적의 기계명령을 사용하는것이다. 이 방법은 간접소비시간을 감소시키지만 바쁜기다림을 사용하기때문에 여전히 효과적이지 못하다.

호상배제를 지원하는 다른 하나의 방법은 조작체계에서 기능들을 주는것이다. 가장 공통적인 수법중에서 두가지가 신호기와 통보문기능이다. 신호기는 프로쎄스사이에서 신호방식을 위하여 쓰이며 호상배제규률을 실시하는데 쉽게 사용할수 있다. 통보문은 호상배제의 실시에서 쓸모 있으며 또한 프로쎄스사이의 통신에서 유효한 수단을 준다.

기본용어

2진신호기	협동루틴	비폐색식
폐색식	교착	신호기
바쁜기다림	통보문넘기기	고갈
병행프로쎄스	감시기	강한 신호기
병행성	호상배제	약한 신호기

복습문제

1. 병행성에서 어느 개념이 의의있는가를 네가지 설계문제점으로 서술하시오..
2. 병행성이 일어 나는 세가지 문맥은 무엇인가?
3. 병행프로쎄스의 집행에서 기본요구는 무엇인가?
4. 프로쎄스사이에서 인식의 세가지 등급을 설명하고 개개를 간단히 정의하시오.
5. 경쟁하는 프로쎄스와 협동하는 프로쎄스사이의 명백한 차이는 무엇인가?
6. 경쟁하는 프로쎄스와 관련되는 세가지 조종문제를 서술하고 개개를 간단히 정의하시오.
7. 호상배제에서의 요구를 설명하시오.
8. 신호기에 기초하여 무슨 조작을 수행할수 있는가?
9. 2진 및 일반신호기사이의 차이는 무엇인가?
10. 강한 및 약한 신호기사이의 차이는 무엇인가?
11. 감시기란 무엇인가?
12. 통보문에서 폐색식과 비폐색식사이의 명백한 차이는 무엇인가?
13. 어떤 조건이 읽기자/쓰기자문제와 일반적으로 련관되여 있는가?

참 고 문 헌

[BEN 82]는 병행성, 호상배제, 신호기 및 다른 련관된 문제들을 매우 명백하고 또 흥미 있게 설명하고 있다. 분산체계를 포함하는 보다 공식적인 취급은 [BEN 90]에서 하고 있다. [AXFO 88]은 읽기 쉽고 쓸모 있는 취급을 하는데 그것은 작성된 풀이를 가지는 많은 문제들을 담고 있다. [RAYN 86]은 쏘프트웨어(례로 Dekker) 및 하드웨어적 방법은 물론 신호기와 통보문을 포괄하여 호상배제를 위한 포괄적이며 명료한 알고리듬의 모임이다. [HOAR 85]는 순차적인 처리와 병행성을 정의하는 공식적인 방법을 보여주는 매우 읽기 쉬운 유명한 책이다. [LAMP 86]은 호상배제에 대한 긴 공식적취급을 하고 있다. [RUDO 90]은 병행성을 리해하는데서 쓸모 있는 방조를 준다. [BACO 98]은 병행성에 대한 구성을 잘하여 취급하고 있다. [BIRR 89]는 병행성을 사용하는 프로그람작성에 대한 좋은 실천적인 안내를 주고 있다. [BUHR 95]는 감시기에 대한 철저한 고찰을 주고 있다. [KANG 98]은 읽기자/쓰기자문제에서의 12개의 서로 다른 일정작성방법들에 대한 교육적인 분석을 주고 있다.

AXFO00 Axford, T. *Concurrent Programming: fundamental Techniques for Real-Time and Parallels Software Design.* New York: Wiley, 1988.
BACO98 Bacon, J. *Concurrent Systems.* Reading, MA:Addison-Wesly, 1998.
BEN82 Ben-Ari, M. *Principles of Concurrent programming.* Englewood Cliffs, NJ:Prentice Hall 1990.
BIRR89 Birrell, A. *An Introducction to Programming with Threads.* SRC Research Report 35, Compaq Systems Research Center, Palo Alto, CA, January 1989. Available at http://www.research. digital. com/ SRC.
BUHR95 Buhr, p., and Foriter, M. "Monitor Classification." *ACM Computing Surveys,* March 1995.
HOAR85 Hoare, C. *Communicating Sequential Processes.* Englewood Cliffs, NJ:Prentice Hall, 1985.
KANG98 Kang, s., and Lee, J."Analysis and Solution of Non-Preemptive Policies for Scheduling Readers and Writers." *Operating Systems Review,* July 1998.
LAMP86 Lamport, L."The mutual Exclusion Problem." *Journal of the ACM,* April 1986.
RAYN86 Yanyal, M. *Algorithms for mutual Exclusion. Cambridge.* MA:MIT Press, 1986.
RUDO90 Rudoph, B. "Self-Assessment Procedure XXI:Concurrency." *Communications of the ACM.* May 1990.

련 습 문 제

1. 프로쎄스와 스레드는 단순한 순차적프로그람들로서 훨씬 더 복잡해 지는 프로그람을 실현하는데서 강력한 구조식도구를 준다. 설명하는데서 교육적인 초기구조는 협동루틴이다. 이 문제의 목적은 협동루틴을 소개하고 그것을 프로쎄스와 비교하는데 있다. 이 단순한 문제를 [CONW 63]으로부터 고찰해 보자. 즉

   80행의 카드를 읽고 그것을 125문자행상에 다음과 같은 변화에 따라 인쇄하시오. 매개 카드의 착공문양뒤에 여분의 공백을 삽입하고 카드에 있는 모든 린접별표쌍(**)을 문자 《↑》로 교체한다.

   ㄱ) 보통 순차프로그람으로 이 문제에 대한 풀이를 전개하시오. 프로그람이 작성하기 까다롭다는것을 알게 될것이다. 프로그람의 여러가지 요소들사이에서의 대화는 80으로부터 125에로의 길이변화 더우기 변환후에 카드착공문양의 길이변화가 2중별표의 출현수에 따라 변하게 되므로 간단하지는 않다. 명료도를 개선하고 잠재적인 오유를 최소화하기 위한 하나의 방법은 응용프로그람을 세개의 분리되여 있는 수속들로 작성하는것이다. 첫 수속이 카드착공문양에서 읽고 공백으로 매개 상을 메우며 문자의 렬을 림시파일에 써넣는다. 모든 카드를 읽은 다음 두번째 수속이 림시파일을 읽고 문자를 대입하고 두번째 림시파일에 써준다. 세번째 수속은 문자렬을 두번째 림시파일에서 읽고 125문자행들을 각각 인쇄한다.

ㄴ) 순차프로그람은 입출력 및 림시파일에 대한 간접소비시간때문에 주의를 끌지 못한다. Conway는 프로그람구조의 새로운 형식인 협동루틴을 제안하였는데 그것은 응용프로그람을 한개의 문자완충기로 련결된 세개의 프로그람으로 작성하도록 한다(그림 5-31). 전통적인 **수속**에서는 피호출 및 호출수속사이에 주인/종속관계가 있다. 호출수속은 수속안의 임의의 점으로부터 호출을 집행할수 있고 피호출수속은 입구점에서 시작되며 호출점에서 호출하는 수속에 돌아 간다. **협동루틴**은 보다 대칭적인 관계를 보여 준다. 매개 호출이 이루어 짐에 따라 피호출수속에서 마지막 능동점으로부터 집행한다. 호출수속이 피호출수속보다 《더 높다》는 그 어떤 의미도 없기때문에 복귀도 없다. 오히려 협동루틴이 개개 지령을 사용하여 다른 협동루틴에 조종을 넘길수 있다. 협동루틴이 기동되는 첫 시기에 그것은 입구점에서 《재개된다.》 다음으로 협동루틴은 그자체의 마지막재개지령점에서 다시 활성화된다. 프로그람에서 한번에 하나의 협동루틴만이 집행상태에 있을수 있으며 이행점이 코드에서 로출된 형식으로 정의되여 있으므로 이것이 병행처리의 실례로는 되지 않는다. 그림 5-31에서 프로그람의 조작을 설명해 보시오.

```
char rsr, sp;
char inbuf[80];
char outbuf[125];
Void read()
{
   while (true)
   {
        READCARD(inbuf);
        For(int i=0; i<80; i++)
        {
            rs=inbuf[I];
            밀어넣기를 재개한다.
        }
        rs="";
        밀어넣기를 재개한다.
   }
}
void print()
{
   while(true)
   {
        for(int j=0; j<125; j++)
        {
                outbuf[j]=sp;
                밀어넣기를 재개한다.
        }
        OUTPUT(outbuf);
   }
}
```

```
void squash()
{
   while (true )
   {
        if(rs!="*")
        {
               sp=rs;
               인쇄를 재개한다.
   }
   else
   {
        RESUME read;
        If(rs= ="*")
        {
               sp="↑";
               인쇄를 재개한다.
        }
        else
        {
               sp="*";
               인쇄를 재개한다.
               Sp=rs;
               인쇄를 재개한다.
        }
   }
   RESUME read;
  }
}
```

그림 5-31. 협동루틴의 응용프로그람

ㄷ) 프로그람은 결속조건을 설명하지 않는다. 입출력루틴 READCARD는 *inbuf*에 80문자의 상을 놓았다면 값 true를 돌려 주고 그렇지 않으면 false를 돌려 준다. 이 사건을 포함하도록 프로그람을 수정하시오. 인쇄된 마지막행이 이런 조건에서 125보다 적은 문자를 가질수 있다는것을 지적해 둔다.

ㄹ) 신호기를 사용하는 세개의 프로쎄스모임으로서 풀이를 다시 작성하시오.

2. 다음과 같이 정의되는 두개의 병행프로쎄스 p와 q를 가지는 병행프로그람을 고찰하자. A, B, C, D, E는 독자적인 원자(나눌수 없는)명령문이다. 주프로그람(보이지 않는)이 두개 프로쎄스의 **동시시작**을 한다.

```
voidp()                         void q ( )
{                               {
   A;                              D;
   B;                              E;
   C;                           }
}
```

선행한 두 프로쎄스의 집행에 대한 가능한 모든 교차처리를 보여 주시오(원자명령문이라는 용어로 집행을 추적하여 이것을 보여 주시오.).

3. 다음의 프로그람을 고찰하자.

```
const int n=50;
int tally;
Void total()
{
      int count;
      for(count=1; count<=n; count++)
      {
               tally++;
      }
}
void main()
{
      tally=0;
      parbegin(total(), total());
      write(tally);
}
```

ㄱ) 이 병행프로그람에 공유된 변수 *tally*의 출력최종값에서 알맞는 아래한계와 웃한계를 확정하시오. 프로쎄스가 어떤 상대적인 속도로 집행할수 있으며 어떤 값은 그것이 개별적인 기계명령에 의해 등록기에 적재된후에만 증가될수 있다고 가정한다.

ㄴ) ㄱ)의 가정밑에서 이 프로쎄스들중의 독자적인 몇개가 병렬로 집행하도록 허락받는다고 가정하자. *tally*의 최종값범위에서 이 수정이 어떤 효과를 가지겠는가?

4. 바쁜기다림이 페색중기다림보다 항상 효과가 적은가(처리기시간을 사용하는것으로 환산하여)를 설명해 보시오..

5. 다음의 프로그람을 고찰하자.

```
boolean bloked[2];
Int turn;
Void p(int id)
{
    while(true )
    {
        blocked[id]= true ;
        while(turn!=id)
        {
            while(blocked[1-id])
             {
                 turn=id;
             }
         }
        /*림계구간*/
        blocked[id]=false;
        /*나머지*/
    }
}
void main()
{
    blocked[o]=false;
    blocked[1]=false;
    turn=0;
    parbegin(P(0), P(1));
}
```

이것은 [HYMA 66]에서 제안된 호상배제문제에 대한 쏘프트웨어적풀이이다. 이 풀이가 부정확하다는것을 보여 주는 대조실례를 찾으시오. *Communications of the ACM*회사조차도 이것에 속히웠다는것을 지적해 두는것은 흥미 있다.

6. Dekker의 알고리듬의 정확성을 증명하시오.

ㄱ) 호상배제가 실시된다는것을 증명하시오. 암시:P_i가 그의 림계구간에 들어 갈 때 다음의 표시가 옳다는것을 보여 주시오. 즉

flag[i] **and** (**not** flag[1-i])

ㄴ) 림계구간에 대한 접근을 요구하는 프로쎄스는 애매하게 지연되지 않는다는 것을 증명하시오. 암시 : 다음의 경우를 고찰하자 ; (1) 단일프로쎄스가 림계구간에 들어 가려고 한다 ; (2) 두개의 프로쎄스가 림계구간에 들어 가려고 하며 (2-ㄱ) *turn*=0이고 *flag*[0]=false, 그리고(2-ㄴ) *turn* =0이고 *flag*[0]=true이다.

7. turn=1-i /*즉 P0은 turn을 1로 설정하고 P1는 turn을 0으로 설정한다.*/로부터 turn=(turn+1)%n /*n=프로쎄스의 수*/까지의 림계구간을 떠날 때 집행되는 명령문을 변화시켜 독자적인 몇개의 프로쎄스에서 작성된 Dekker의 알고리듬을 고찰하자. 병행하여 집행하는 프로쎄스의 수가 둘이상일 때 알고리듬을 평가하시오.

8. 피터슨의 알고리듬을 *N*개의 프로쎄스사이에서 호상배제를 보장하도록 일반화할 수 있다. 두개의 전역배렬을 *q*와 *turn*이라고 가정하자. *q*의 *N*개 요소 및 *turn*

의 N-1개요소의 초기값은 모두 0이다. 매개 프로세스는 배렬첨수로 사용되는 전용변수 j와 k를 가지고 있다. 프로세스 i에서의 알고리듬은 다음과 같다.

```
global integer arrays q[N], turn[N-1]

    do
1  for  ( int  j = 1;  j <>= N-1;  j++ )
2  {
3     q [i] = j;
4     turn [j] = i;
5     {
6          for  ( int  k = 1;  k <>= N;  k++ )
7          {
8              if  ( k ! = i )
9                     k++;
10         }
11
12    while((q[k]>=j)&&(turn[j]==i));
          /*프로세스 i의 림계구간*/
13  q [I ]= 0;
                 /*프로세스 i의 나머지구역*/
        while(true );
```

국부변수 j의 값을 프로세스 i가 집행하는 알고리듬의 《단계》로 생각하는것이 편리하다. 전역변수 q는 매개 프로세스의 단계를 가리킨다. 프로세스가 림계국면에 들어갈 때 그것은 단계 N에로 넘어 간다(명령문 $q[i]=N$가 이것을 보장한다. 실제적으로 이 명령문은 단지 증명에서 사용되는 언어를 간단화하기 위한 것이며 알고리듬의 정확성을 위해서는 필요 없다.).

만일 $q[x] > q[y]$이면 프로세스 x가 프로세스 y를 선행한다(즉 앞에 있다.). 이 알고리듬이

호상배제를 하게 하며
교착이 없게 하며
그 어떤 고갈도 주지 않는다

는것을 보여 주려고 한다.

그렇게 하기 위해 다음과 같은 명제들을 증명하시오

ㄱ) 명제 1 : 모든 다른 프로세스를 선행하는 프로세스가 적어도 하나의 단계를 전진할수 있다. 암시 : 프로세스 i가 4행을 시작할 때

$$j = q[i] > q[k], \quad k \neq I$$를 알게 하시오.

ㄴ) 명제 2 : 프로세스가 단계 j로부터 단계j+1로 넘어 갈 때 다음의 요구들중의 하나는 정확히 유지한다. 즉

- 그것이 모든 다른 프로세스들을 선행한다. 즉
- 그것이 단계 j에 홀로 있지 않는다.

암시 : 프로세스 i가 $q[i]$를 막 전진시키도록 하고 등식(1)이 참인가 아닌가를 고찰하시오.

ㄷ) 명제 3: 단계 j에 (적어도) 두개의 프로쎄스가 있다면 (적어도) 하나의 프로쎄스가 매개 단계 k $(1 \le k \le j-1)$에 있다.

암시 : j를 받아 들여 증명하시오.

ㄹ) 명제 4 : 단계 j에 있을수 있는 프로쎄스의 최대수는 $N-j+1(1 \le j \le N-1)$이다.

암시 : 이것은 간단한 연산이다.

ㅁ) 명제 1~4에 기초하여 알고리듬이 호상배제, 교착해제를 주며 아무런 고갈도 주지 않는다는것을 보여 주시오.

9. 호상배제를 위한 또하나의 쏘프트웨어적방법은 Lamport의 **빵집알고리듬**[LAMP 74]인데 그렇게 부르는것은 그것이 모든 손님들이 도착하는 차례로 번호가 붙은 표를 받고 매개 사람들을 차례로 봉사하도록 하는 빵집들과 다른 상점들에서의 실천에 기초하고 있기때문이다. 그 알고리듬은 다음과 같다

```
boolean choosing[n];
int number[n];
while(true)
{
    choosing[i]= true ;
    number[i]=1+getmax(number[],n);
    choosing[i]=false;
    for (int j=0; j<>n; j++)
    {
        while (choosing[j])
        {};
        while((number[j]!=0)&&(number[j],j)<>(number[i],i))
        {};
        /*림계구간*/
        number[i]=0;
        /*나머지*/
    }
}
```

배렬 *choosing*과 *number*는 false 및 0으로 각각 초기화된다.
매개 배렬의 i번째 요소를 프로쎄스 i가 읽기 및 쓰기할수 있지만 다른 프로쎄스는 읽기만 할수 있다.
표시 $(a, b)<(c, d)$를 $(a<c)$ 또는 $(a=c$ 및 $b<d)$로 정의한다.

ㄱ) 알고리듬을 단어로 서술하시오

ㄴ) 이 알고리듬이 교착을 피한다는것을 보여 주시오

ㄷ) 그것이 호상배제를 실시한다는것을 보여 주시오.

10. 호상배제에 대한 다음의 쏘프트웨어적방법이 기억기접근준위에서 기본적인 호상배제에 의존하지 않는다는것을 론증하시오.

ㄱ) 빵집알고리듬

ㄴ) 피터슨의 알고리듬

11. 그림 5-5의 방식에 호상배제를 주기 위하여 특수한 기계명령을 사용할 때 프로쎄스가 림계구간에 대한 접근을 허용 받기전에 얼마나 오래 기다려야 하는가 하는데 대한 그 어떤 조종도 없다. 검사설정명령을 사용하지만 림계구간에 들어

가기 위하여 기다리고 있는 임의의 프로쎄스가 n-1개의 차례에 그렇게 하리라고 보는 알고리듬을 작성하시오. 여기서 n은 림계구간에 대한 접근을 요구할수 있는 프로쎄스의 수이며 《차례》는 림계구간을 떠나는 하나의 프로쎄스와 접근을 허용 받는 또다른 프로쎄스로 구성되는 어떤 사건이다.

12. 신호기에 대한 다음의 정의를 고찰하자

```
void wait(s)
{
   if (s.couint>0)
   {
       s.count-;
   }
   else
   {
       프로쎄스를 s.queue에 배치하고 페색시킨다.
   }
}
void signal(s)
{
   if(신호기 S에서 중단된 적어도 하나의 프로쎄스가 있다.)
   {
          프로쎄스 p를  s.queue에서 꺼낸다;
          프로쎄스 P를 준비목록에 넣는다;
   }
   else
          s.count++;
}
```

이 정의들의 모임을 그림 5-6의 정의들의 모임과 비교하시오. 하나의 차이를 주목하시오. 즉 선행한 정의에서 신호기는 절대로 부의값을 가질수 없다. 프로그람에서 사용할 때 두개의 정의들의 모임의 효과에서 어떤 차이가 있는가? 즉 프로그람의 의미를 바꾸지 않고 하나의 모임이 다른 모임을 대신할수 있는가?

13. 2진신호기를 사용하여 일반신호기를 실현할수 있다. 조작 waitB와 signalB 그리고 두개의 2진신호기, 지연 및 mutex를 사용할수 있다. 다음의것을 고찰하자. 즉

```
void Wait(semaphore s)
{
    WaitB(mutex);
    s--;
    if(s<>0)
    {
        SignalB(mutex);
        WaitB9delay);
    }
    else
       SignalB(mutex);
}

void Signal(semaphore s)
{
```

```
        WaitB(mutex);
        S++;
        If (s<=0)
            SignalB(delay);
      SignalB(mutex);
    }
```

초기에 *s*는 요구하는 신호기의 값모임이다. 매개 Wait조작은 S를 감소시키며 매개 Signal조작은 S를 증가시킨다. 2진신호기 mutex는 1로 초기화되여 있는데 *s*를 갱신하는데서 호상배제가 있다는것은 보증한다. 2진신호기 delay는 0으로 초기화되는데 프로쎄스를 중단시키기 위하여 사용된다. 선행한 프로그람에는 결함이 있다. 결함을 론증하고 그것을 고칠 변경안을 제안하시오. 암시 : S가 초기에 0일 때 두개의 프로쎄스가 각각 Wait(s)를 호출하여 첫 호출이 방금 SignalB(mutex)를 수행하였지만 WaitB(delay)를 수행하지 못한 다음에 Wait(s)에 대한 두번째 호출이 같은 점에 착수한다고 가정하자. 할 필요가 있는 모든것은 프로그람의 단일한 행에 옮겨 진다.

14. 1978년에 딕스트라는 유한개의 약한 신호기를 사용하여 알려 지지 않았지만 유한개의 프로쎄스에 적용할수 있는 고갈을 피하는 호상배제에 대한 그 어떤 풀이도 없다는 가설을 내놓았다. 1979년에 J.M.Morris는 세개의 약한 신호기를 사용하는 알고리듬을 출판하여 이 가설을 론박하였다. 그 알고리듬의 동작은 다음과 같이 서술할수 있다. 즉 하나 또는 몇개의 프로쎄스가 wait(S)조작에서 기다리고 있으며 또다른 프로쎄스는 signal(S)를 집행하고 있고 신호기 S의 값은 수정되지 않으며 기다리는 프로쎄스들중에서 하나는 wait(S)에 무관계하게 폐색되지 않는다. 세개의 신호기와 별개로 알고리듬은 두개의 부아닌 옹근수변수를 알고리듬의 일정한 구역에 있는 프로쎄스의 수에 대한 계수기로 사용한다. 그러므로 신호기 A와 B는 1로 초기화되며 한편 신호기 M과 계수기 NA 및 NM은 0으로 초기화된다. 호상배제의 신호기 B는 공유변수 NA에 대한 접근을 막아 준다. 자기의 림계구간에 들어 가려고 하는 프로쎄스는 신호기 A와 M으로 표현된 두개의 장벽을 넘어야 한다. 계수기 NA와 NM은 각각 장벽 A를 넘을 준비가 된 몇개의 프로쎄스와 이미 장벽 A를 넘었지만 아직 장벽 M을 넘지 못한 프로쎄스들을 가지고 있다. 규약의 두번째 부분에서 M에서 폐색된 NM의 프로쎄스들은 첫 부분에서 사용한것과 류사한 종속련결수법을 사용하여 하나씩 자기들의 림계구간에 들어 갈것이다. 이 설명에 따르는 알고리듬을 정의하시오.

15. 다음 문제가 시험에 한번 사용되였다.
유라시아공원은 공룡박물관과 사냥놀이를 위한 공원으로 되여 있다. m명의 려행자와 n개의 1인용려행자차가 있다. 려행자들은 박물관을 잠간 돌아 본다음 사냥용차에 오르기 위하여 줄을 선다. 차를 사용할수 있을 때 그것이 유지할수 있는 한명의 려행자를 태우고 우연시간동안 공원을 돌아 다닌다. n개의 차가 모두 려행자를 태우고 나가 있으면 차를 타려는 려행자는 기다리며 만일 차가 태울 준비가 되여 있지만 기다리는 려행자가 한명도 없으면 그 차가 기다린다. 신호기를 사용하여 m명의 려행자프로쎄스와 n개의 차프로쎄스를 동기화시키시오.

다음의 골격코드는 시험장바닥의 종이쪼박지상에서 발견되였다. 정확성의 견지에서 그것을 평가하시오. 문법이나 빠진 변수선언은 무시하시오. P와 V는 wait와 signal에 대응한다는것을 기억하시오

```
\ begin{verbatim}
resource Jurassic_Park()
 sem car_avail:=0, car_taken:=0,car_filled:=0,

     passenger_released:=0

process passenger(I:=1 to num_passengers)
  do true -> nap(int(random(1000*wander_time)))
    p(car_avail); V(car_traken); P(car_filled)
   p(passenger_repleased)
  od
end passenger

process car(j:=1 to num_cars)
  do true ->V(car_avail);P(car_taken);V(car_filled)
   nap(int(random(1000*ride_time)))
   V(passenger_released)
  od
end car
end Jurassic_Park
 \ end{verbatim}
```

16. 그림 5-12의 주해와 표 5-2에서 《교착을 일으킬수 있기때문에 조건명령문을 소비자의 림계구간(s가 조종하는)안에 이동시키는것은 간단하게 할수 없을것이다.》라는것을 설명하였다. 표 5-2와 류사한 표를 사용하여 이것을 설명하시오.

17. 그림 5-13에서 정의된 무한완충기식의 생산자/소비자문제에 대한 풀이를 고찰하자. 생산자와 소비자가 같은 속도에서 실행하고 있는(일반적인) 경우를 가정하자. 방안은 다음과 같이 될수 있다. 즉

생산자 : 추가 ; 신호 ; 생산 ; … ; 추가 ; 신호 ; 생산 ; …
소비자 : 소비 ; … ; 꺼내기 ; 기다림 ; 소비 ; … ; 꺼내기 ; 기다림 ; …

생산자는 항상 소비자가 이전 요소를 소비하는 동안에 완충기에 새로운 요소를 추가하고 신호를 보내려고 한다. 생산자는 항상 빈 완충기에 추가하려고 하며 소비자는 항상 완충기에서 단독적인 항목을 꺼내려고 한다. 소비자가 신호기에서 절대로 폐색되지 않는다고 하여도 신호기수법에 대한 많은 호출이 이루어져 상당한 간접소비시간을 초래한다. 이 환경에서 보다 효과적으로 될 새로운 프로그람을 구성하시오. 암시 : n이 값 –1을 가지도록 하자. 이것은 완충기가 비여 있을뿐아니라 소비자가 이 사실을 검출하였고 생산자가 새 자료를 줄 때까지 폐색하려고 한다는것을 의미한다. 풀이는 그림 5-13에서 국부변수 m을 사용하지 말아야 한다.

18. 그림 5-16을 고찰하자. 다음의것을 서로 교체하면 프로그람의 의미가 변하겠는가?

ㄱ) wait(e) ; wait(s)
ㄴ) signal(s) ; signal(n)
ㄷ) wait(n) ; wait(s)
ㄹ) signal(s) ; signal(e)

19. 유한완충기를 가진 생산자/소비자문제의 설명에서 한 정의가 완충기에서 기껏 n-1개의 입구를 허용한다는데 주목하자.

ㄱ) 이것은 무엇때문인가?
ㄴ) 이 결함을 퇴치하도록 알고리듬을 수정하시오.

20. 공평한 리발소와 관련되는 다음의 질문들에 대답하시오(그림 5-20).

ㄱ) 코드는 손님의 리발을 끝낸 리발사가 손님의 지불을 받아 들일것을 요구하는가?
ㄴ) 리발사가 항상 같은 리발의자를 사용하는가?

21. 그림 5-20의 공평한 리발소에서 몇가지 문제가 남아 있다. 다음의 문제들을 정정하도록 프로그람을 수정하시오.

ㄱ) 출납은 한 손님으로부터 지불을 접수할수 있으며 두명 또는 그이상이 지불하려고 기다리고 있으면 다른 사람들을 해방시킬수 있다. 다행히 일단 손님이 지불을 하면, 지불은 꼭 하게 되는데, 그래서 마침내 정확한 돈액수가 출납등록기에 떨어 진다. 그럼에도 불구하고 손님이 지불하자마자 정확한 손님을 해방하도록 할수 있다.

ㄴ) 신호기 *leave_b_chair* 는 추측하건대 단일한 리발의자에 대한 여러개의 접근을 막아 준다. 불행하게도 이 신호기는 모든 경우에 성공하지 못한다. 실례로 세명의 리발사가 모두 리발을 끝냈고 *wait(leave_b_chair)*에서 페색된다고 가정하자. 손님들중의 두사람이 리발의자를 떠나기 직전에 중단된 상태에 있다. 세번째 손님이 그의 의자를 떠나 *signal(leave_b_chair)* 를 집행한다. 어느 리발사가 해방되는가? *leave_b_chair*대기렬이 선입선출이므로 페색된 첫 리발사가 해방된다. 그가 신호하는 손님의 머리를 깎고 있던 리발사인가? 그럴수도 있고 아닐수도 있다. 만일 아니라면 새 손님이 따라 와서 방금 막 일어 나려고 하는 손님의 무릎에 올라 앉게 될것이다.

ㄷ) 프로그람은 리발의자가 비여 있을 때에도 손님이 의자에 앉아 있을것을 요구한다. 당연하지만 이것은 덜 중요한 문제이며 그것을 고치는것은 이미 좀 불결한 코드를 더 복잡하게 한다. 그래도 그것을 해보시오.

22. 이 문제는 신호기를 사용하여 세가지 형태의 프로쎄스에 대한 조종을 설명한다.[4]싼타클루스는 북극에 있는 자기의 상점에서 자며 다만 (1) 남태평양에서 려행을 마치고 돌아 오는 아홉마리의 북극사슴 모두라든지 또는 장난감들을 만드는데서 난관에 부딪치는 일부 꼬마요정들이 깨울수 있으며 쌘터를 좀 재우기 위하여 꼬마요정들은 그들중에서 셋이 문제를 가질 때에만 깨울수 있다. 세 꼬마요정이 자기들의 문제를 풀고 있을 때 쌘터를 만나고 싶어 하는 다른 꼬마요정들은 그 꼬마요정들이 돌아가기를 기다려야 한다. 만일 쌘터가 깨여 나서 자기의 상점문어구에서 기다리고 있는 세 꼬마요정을 발견하면 열대지방에서 돌아 오는 마지막북극사슴과 함께 쌘터는 꼬마요정들이 크리스마스 다음날까지 기다릴수 있다고 결심을 내리는데 그것은 보다 중요한것이 준비된 썰매를 타고 려행하는것이기때문이다(북극사슴이 열대지방을 떠나고 싶어 하지 않으며 따라서 그것들은 마지막 가능한 순간까지 거기에 머물러 있는다고 가정한다.). 돌아 오는 마지막 북극사슴은 다른 북극사슴들이 따뜻한 집에서 기다리고 있는 동안에 쌘터를 모셔 온다음 썰매에 마구를 메워야 한다. 신호기를 가지고 이 문제를 푸시오.

23. 통보문넘기기와 신호기가 다음의것들로 등가적인 기능을 가진다는것을 보여 주시오.

ㄱ) 신호기를 사용하는 통보문넘기기의 실현.

암시: 매개가 통보문슬로트로 구성되여 있는 우편통들을 유지하기 위해 공유완충기구역을 사용하도록 하시오.

ㄴ) 통보문넘기기를 사용하는 신호기의 실현.

암시 : 개별적인 동기식프로쎄스를 도입하시오.

---

[4]. 버몬트에 있는 쎄인트 마이켈대학의 존 트로노가 이 문제를 내놓았다.

제 6 장. 병행성: 교착과 고갈

이 장에서는 병행성에 대한 고갈의 계속으로 병행처리를 지원하는데서 애로로 제기되는 두가지 문제 즉 교착과 고갈에 대하여 서술한다. 먼저 교착의 기초원리와 고갈과 관련되는 문제를 설명한다. 다음에 교착을 취급하는 세가지 일반적인 방법인 예방, 검출, 피하기에 대하여 설명한다. 계속하여 동기화와 교착문제를 설명하는데 리용되는 한가지 고전적인 문제와 철학자식사문제를 고찰한다.

제5장과 이 장에서는 병행성과 교착을 하나의 옹근체계로서는 고찰하지 못하고 제한된 범위에서 취급한다.

제 1 절. 교착의 원리

교착은 체계자원들을 경쟁적으로 리용하거나 서로 통신하는 프로쎄스모임의 영구적인 페색이라고 정의할수 있다. 이것은 병행프로쎄스관리에서 제기되는 다른 문제들과는 달리 일반적인 경우에 해결방법이 없다.

모든 교착들은 두개 또는 그이상의 프로쎄스들에 의해 자원들이 충돌하는 경우를 포함한다. 일반적인 실례로서 교통흐름의 교착을 들수 있다. 그림 6-1 ㄱ에서는 4대의 자동차가 거의 동일한 시각에 네거리교차점에 도착한 상황을 보여 주고 있다. 교차점의 4개의 1/4구역은 조종을 필요로 하는 자원이다. 특히 4대의 자동차가 교차점을 통과하여 곧바로 가려고 한다면 자원에 대한 요구가 다음과 같이 제기된다.

- 북쪽으로 달리는 자동차에는 1/4구역인 1과 2가 필요하다.
- 서쪽으로 달리는 자동차에는 1/4구역인 2와 3이 필요하다.

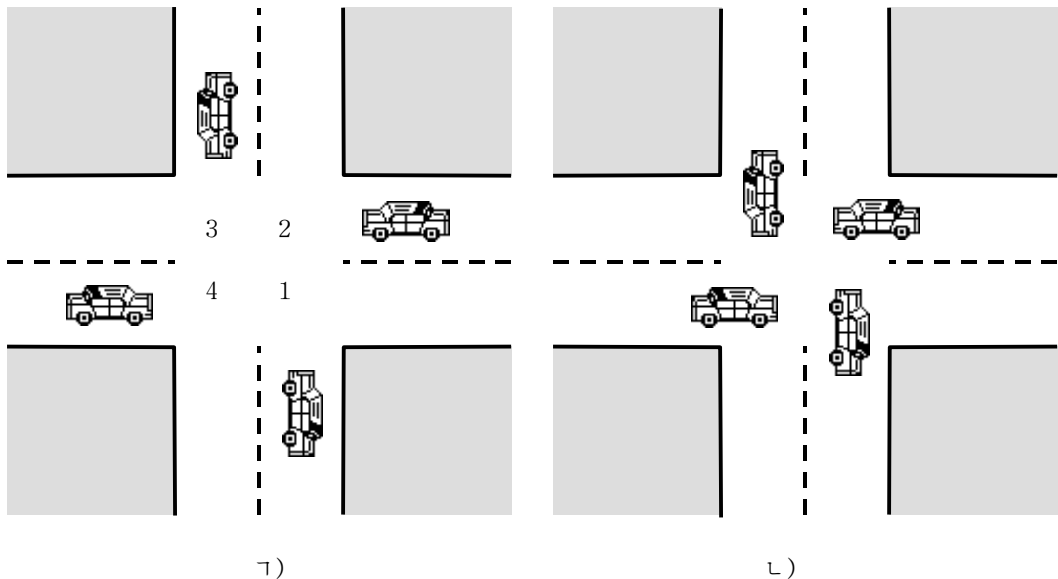



그림 6-1. 교착의 설명
ㄱ-교착의 가능성, ㄴ-교착

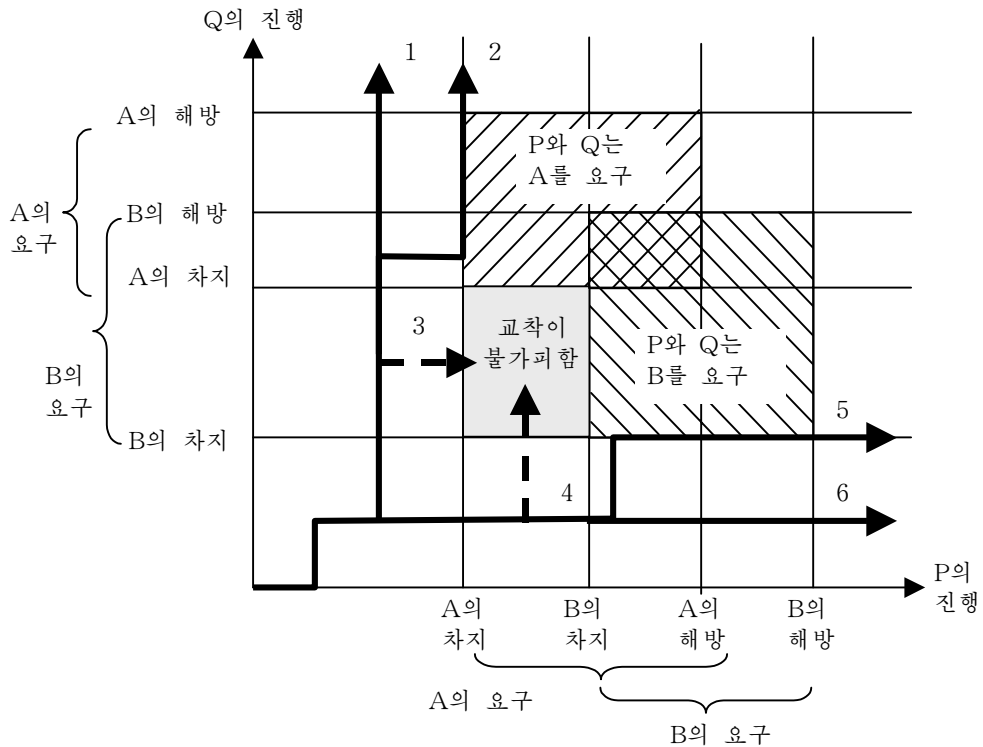



그림 6-2. 교착의 실례 [BACO 98]

- 남쪽으로 달리는 자동차에는 1/4구역인 3과 4가 필요하다.
- 동쪽으로 달리는 자동차에는 1/4구역인 4와 1이 필요하다.

이 규칙은 두대 또는 석대의 자동차가 교차점에 있을 때에만 성립한다. 실례로 북쪽경계와 서쪽경계의 자동차만이 교차점에 도착한다면 북쪽경계의 자동차는 기다리게 되고 서쪽경계의 자동차는 전진한다. 그러나 만일 4대의 자동차가 모두 거의 동일한 시각에 도착한다고 하면 매개 자동차가 한개의 자원(구역)을 차지하지만 교차점에 들어 서서부터 멈춰 서게 되고 결국 교착상태가 발생된다. 만일 4대의 자동차가 모두 통행규칙을 무시하고 교차점에로 전진(주의!) 한다면 매개 자동차는 한개의 자원(1/4 구역)을 차지하지만 필요한 두번째 자원이 다른 자동차에 대해 이미 차지되고 있으므로 전진할수 없다. 그리하여 다시 교착상태에 빠진다.

이제 프로쎄스와 콤퓨터자원들을 포함하는 교착에 대하여 서술한 실례를 보자. 그림 6-2에서는 두개의 자원에 대하여 경쟁하는 두개의 프로쎄스의 진행정형을 보여 주고 있다. 매개 프로쎄스는 어떤 시간주기동안 두가지 자원을 배타적으로 사용한다. 프로쎄스 P는 다음의 일반형식을 가진다.

Process P
⋮
A의 차지
⋮
B의 차지

⋮

A의 해방

⋮

B의 해방

⋮

또한 프로세스 Q는 다음의 일반형식을 가진다.

Process Q

⋮

B의 차지

⋮

A의 차지

⋮

B의 해방

⋮

A의 해방

⋮

그림 6-2에서 x축은 P의 집행정형을 표시하고 y축은 Q의 집행정형을 표시한다. 따라서 두 프로세스의 공동진행정형은 원점으로부터 북동방향으로 전진하는 경로로 표시된다. 단일처리기인 경우에는 어떤 순간에 한개의 프로세스만 집행할수 있다. 이때 경로는 서로 엇바뀌는 수평 또는 수직구간으로 이루어 지는데 수평구간은 P가 집행되고 Q가 기다리는 주기를, 수직구간은 Q가 집행되고 P가 기다리는 주기를 각각 표시한다.

그림에서는 6개의 각이한 집행경로를 보여 주고 있다, 이것은 다음과 같이 종합할수 있다. 즉

1. Q는 B와 A를 차지하고 다음 B와 A를 해방한다. P가 다시 집행을 시작할 때 그것은 두가지 자원을 차지할수 있다.
2. Q는 B와 A를 차지한다. P는 집행되다가 A를 요청할 때 페색된다. Q는 B와 A를 해방한다. P가 다시 시작될 때 그것은 두가지 자원을 차지할수 있다.
3. Q는 B를 차지하고 다음에 P는 A를 차지한다. 집행이 계속될 때 Q는 A에 의해 페색되고 P는 B에 의해 페색되므로 교착은 불가피하다.
4. P는 A를 차지하고 다음 Q는 B를 차지한다. 집행이 계속될 때 Q는 A에 의해 페색되고 P는 B에 의해 페색되므로 교착은 불가피하다.
5. P는 A와 B를 차지한다. Q는 집행되고 B에 대한 요구에 의해 페색된다. P는 A와 B를 해방한다. Q가 다시 집행될 때 그것은 두가지 자원을 차지할수 있다.
6. P는 A와 B를 차지하고 다음 A와 B를 해방한다. Q가 다시 집행될 때 그것은 두가지 자원을 차지할수 있다.

교착이 발생되는가 발생되지 않는가 하는것은 집행의 동적상태와 응용의 세부에 관계된다. 실례로 P가 두가지 자원을 동시에 요구하지 않는다면 다음의 형식을 가지게 된다. 즉

Process P

⋮

A의 차지

⋮
A의 해방
⋮
B의 차지
⋮
B의 해방
⋮

이러한 정황을 그림 6-3에 반영하고 있다. 두 프로세스의 상대적인 동기에 관계되지 않는 방법을 쓰게 되면 교착은 발생할수 없다.

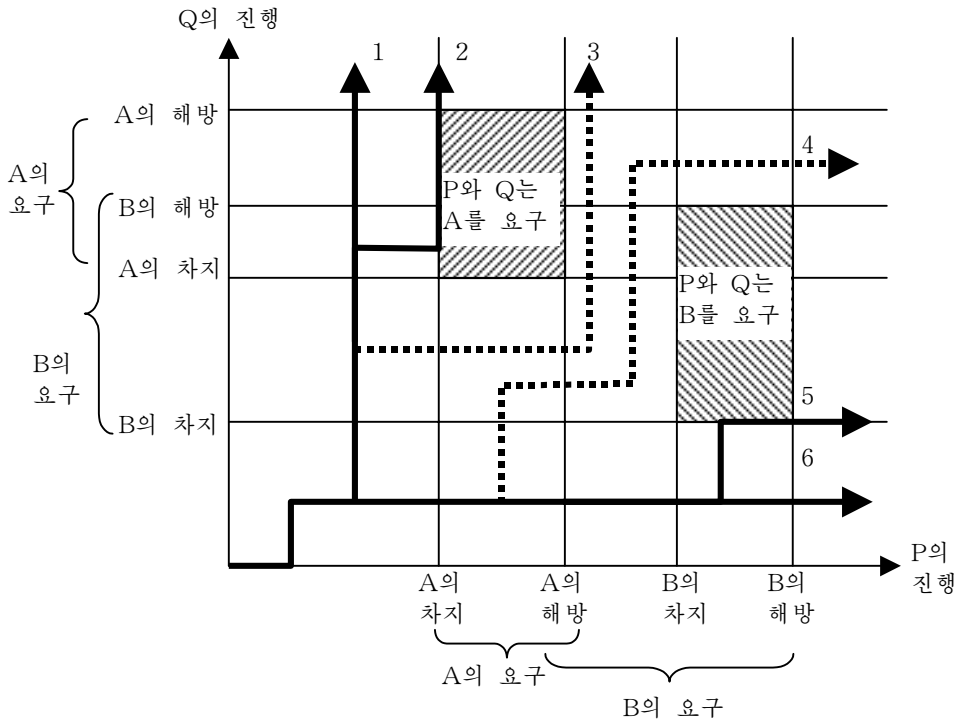



그림 6-3. 비교착의 실례[BACO 98]

재사용가능한 자원

일반적으로 자원은 두가지 부류 즉 재사용가능한 자원과 소비되는 자원으로 분류할수 있다. 재사용가능한 자원은 한번에 한개 프로세스에 의해서만 사용할수 있고 또 사용으로 소멸되지 않는 자원을 말한다. 프로쎄스들은 자원장치들을 획득하는데 그것들을 다른 프로쎄스들에서 다시 사용하기 위해 후에 해방한다. 재사용가능한 자원들의 실례로서 처리기들, 입출력통로들, 주기억기와 2차기억기들, 장치들 그리고 파일, 자료기지, 신호기들과 같은 자료구조들을 들수 있다.

재사용가능한 자원들을 포함하는 교착의 실례로서 디스크파일 D와 테프구동기 T에 배타적으로 접근하기 위해 경쟁하는 두개의 프로쎄스를 고찰하자. 프로그람들은 그림

6-4에 서술한 조작에 참가한다. 매개 프로쎄스가 한개의 자원을 차지하고 다른 자원을 요청한다면 교착이 발생한다. 실례로 다중프로그람처리체계가 두개의 프로쎄스를 다음과 같이 교대로 집행한다면 교착이 발생한다. 즉

p0p1q0q1p2q2

이것은 조작체계의 설계자에 관한 문제라기보다 프로그람작성상 오유인듯이 보인다. 그러나 여기서는 병행프로그람설계를 취급하지 않는다. 그러한 교착들은 발생할 때 흔히 복잡한 프로그람론리속에 내장되여 있으므로 검출하기가 곤난하다. 그러한 교착을 취급하는 한가지 방책은 자원들을 요청하는 순서에 관한 제약조건들을 체계설계에 반영하는것이다.

재사용가능한 자원을 가진 교착의 다른 실례는 주기억기에 대한 요청을 처리해야 하는 경우이다. 기억기할당에 쓸수 있는 공간이 200Kbyte이고 다음의 순서로 요청이 발생한다고 하자.

P1	P2
⋮	⋮
Request 80 Kbytes;	Request 70 Kbytes;
⋮	⋮
Request 60 Kbytes;	Request 80 Kbytes;

프로쎄스 P

걸음	작업
p_0	Request( ))
p_1	Lock(D)(
p_2	Request(`)
p_3	Lock(T)
p_4	기능수행
p_5	Unlock(I)
p_6	Unlock(T)

프로쎄스 Q

걸음	작업
q_0	Request(`)
q_1	Lock(T)
q_2	Request( ))
q_3	Lock(D)
q_4	기능수행
q_5	Unlock(T)
q_6	Unlock(I)

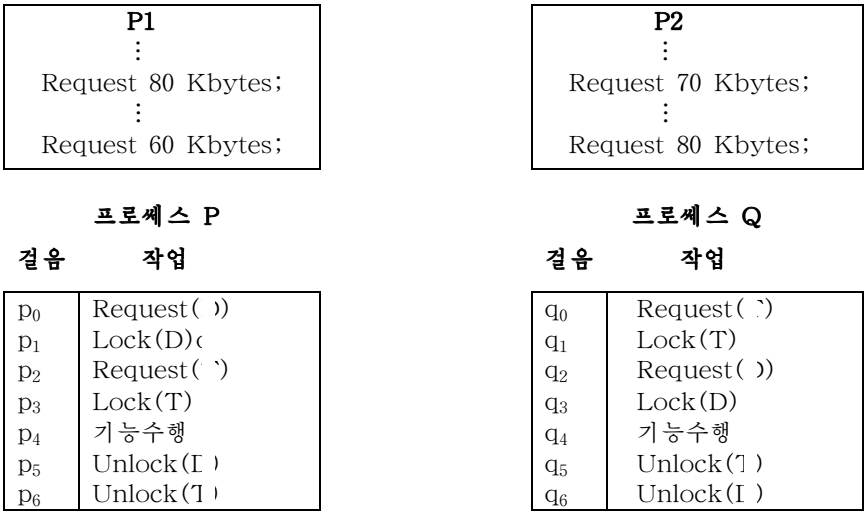



그림 6-4. 재사용가능한 자원들에 대하여 경쟁하는 두개의 프로쎄스실례

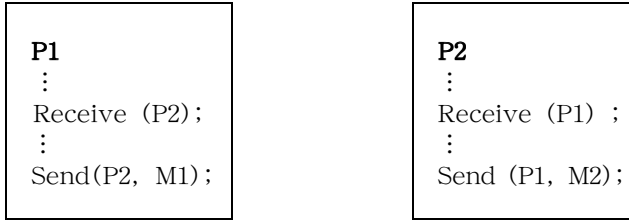
만일 두개의 프로쎄스가 두번째 요청을 내는데로 전진한다면 교착이 발생한다. 요청하는 기억기량이 사전에 알려 지지 않는다면 체계설계의 제약조건들에 의하여 이러한 형태의 교착은 처리하기가 곤난해 진다. 이러한 특정한 문제를 처리하는 제일 좋은 방도는 제8장에서 취급하는 가상기억기를 사용하여 그 가능성을 제거해 버리는것이다.

소비되는 자원

소비되는 자원은 창조(발생)할수도 있고 파괴(소실)될수도 있는 자원을 말한다. 대표적으로 특정한 형태의 소비되는 자원수에는 제한이 없다. 발생되는 비페색된 프로쎄스는 임의의 개수의 자원을 해방할수 있다. 프로쎄스가 자원을 차지하면 그 자원은 소비된다. 소비자원의 실례로서는 새치기들, 신호들, 통보문들과 입출력완충기의 정보를 들수 있다.

소비되는 자원들을 포함하는 교착의 실례로서 다음의 프로쎄스쌍을 고찰하자. 여기서 매개 프로쎄스는 다른 프로쎄스로부터 통보문을 수신한 다음에 통보문을 다른 프로쎄스에 보내려고 한다. 즉

```
P1
 ⋮
Receive (P2);
 ⋮
Send(P2, M1);
```

```
P2
 ⋮
Receive (P1) ;
 ⋮
Send (P1, M2);
```

만일 Receive가 페색중(즉 수신하고 있는 프로쎄스는 통보문이 수신될 때까지 페색된다)에 있으면 교착이 발생한다. 교착이 발생하였다는것은 설계에 오유가 있기때문이다. 그러한 오유들은 극히 미묘하여 검출하기가 어려울수 있다. 더우기 사건들이 묘하게 조합되여 교착을 발생시킬수도 있다. 그러므로 문제가 제기되는 프로그람을 일정한 기간 지어 몇년간 사용해 보면서 검토해 볼수도 있다.

모든 형태의 교착들을 처리할수 있는 완전히 효과적인 방책은 없다. 이미 개발된 가장 중요한 방법들의 기본요소들인 검출, 예방 및 피하기를 요약하여 표 6-1에 제시하였다. 여기서는 교착을 발생시키는 조건들을 밝힌 다음에 이 결과들을 차례로 검토하기로 한다.

교착을 발생시키는 조건

발생할수 있는 교착에 대하여 다음의 세가지 조건을 제시해 주어야 한다.

1. **호상배제** 한개의 프로쎄스는 한번에 한개의 자원을 유일하게 사용할수 있다.
2. **유지 및 기다림** 프로쎄스는 할당된 자원을 다른것의 할당을 기다리는 동안 유지할수 있다.
3. **비선취권** 자원은 그것을 유지하고 있는 프로쎄스로부터 강제로 제거될수 없다.

이 여러가지 조건들은 반드시 필요한것들이다. 실례로 호상배제는 결과들의 일관성과 자료기지의 완정성을 확고히 보장하는데 필요하다. 마찬가지로 선취권을 제멋대로 수행할수 없으며 특히 자료자원들이 요구될 때에는 재연산회복기구에 의하여 안받침되여야 한다. 재연산회복기구는 되풀이할수 있는데 충분한 앞선 상태에로 프로쎄스와 그것의 자원들을 회복시킨다.

교착은 이 세가지 조건에 의해 존재할수도 있고 존재하지 않을수도 있다. 교착이 실제적으로 발생하자면 네번째 조건이 만족되여야 한다.

4. **순환기다림** 련쇄에서 매개 프로쎄스가 다음 프로쎄스에 필요되는 적어도 한개의 자원을 유지하는것과 같은 닫긴 프로쎄스련쇄가 존재한다(그림 6-5).

첫 세가지 조건들은 필요는 하지만 교착이 존재하는데 충분하지는 못하다. 실제적으로 네번째 조건은 첫 세가지의 조건의 잠재적인 결과이다.첫 세가지의 조건이 존재할 때 어떤 사건순서렬이 발생할수 있는데 이것은 해결할수 없는 순환기다림에로 이끌어 나간다. 해결할수 없는 순환기다림은 사실상 교착을 의미한다. 조건 4로서 작성된 순환기다림은 첫 세가지의 조건이 유지되므로 해결할수 없는 순환기다림이다.따라서 우의 네가지 조건이 바로 교착이 발생하기 위한 필요하고도 충분한 조건으로 된다.[1]

---

[1] 사실상 모든 교과서들에서는 단순히 이 네가지 조건을 교착에 필요한 조건으로 지적하고 있지만 그러한 표현은 일부 미묘한 문제를 모호하게 한다[SHUB 90]. 순환기다림조건인 항목 4는 다른 세가지 조건들과 근본적으로 다르다. 항목 1～3은 방법상의 해결문제와 관련된 항목이라면 항목 4는 요청순서렬에 의하여 발생하고 관계되는 프로쎄스에 의하여 해방할수 있게 하는 환경항목이다. 순환기다림을 세가지의 필요조건과 결부시키면 예방과 피하기사이의 구별을 정확히 할수 없게 한다.

표 6-1. 조작체계에서 교착의 검출, 예방 및 피하기방법의 요약[ISLO 80]

원리	자원할당방법	각이한 방안	중요한 우점	중요한 결함
예방	보존물; 자원들을 불충분하게 넘겨준다.	모든자원을 동시에 요청	● 한무리의 동작을 수행하는 프로쎄스들에 잘 맞는다. ● 선취권이 필요 없다.	● 비효률적이다. ● 프로쎄스의 시동을 지연시킨다. ● 앞으로의 자원요구조건을 알아야 한다.
		선취권	● 상태를 보관하고 쉽게 회복할 수 있는 자원에 응용할 때 편리하다.	● 필요이상 더 자주 선취한다. ● 주기적인 재시동을 당하기 쉽다.
		자원순서짓기	● 콤파일시간검사들을 거쳐 시행하는것이 편리하다. ● 문제가 체계설계에서 해결되므로 실행시간계산이 필요하다.	● 많은 사용이 없이 선취한다. ● 증대되는 자원요청들은 허용하지 않는다.
피하기	검출과 예방사이의 중간	적어도 한개의 안전한 경로를 찾기 위해 조작	● 선취권이 필요하다.	● 앞으로 자원요구들을 알아야 한다. ● 프로쎄스들이 오랜시간 폐색될수 있다.
검출	대단히 공평; 요청된 자원들을 필요한곳에 준다.	주기적으로 교착 시험을 요구	● 프로쎄스의 초기화의 지연이 없다. ● 직결처리를 헐하게 한다.	● 고유한 선취권이 상실된다.

제 2 절. 교착의 예방

교착을 예방하는 방책은 한마디로 교착가능성을 배제하는 방법으로 체계를 설계하는것이다. 교착예방방법은 두가지 부류로 갈라 볼수 있다. 간접적인 교착예방방법은 우에서 지적한 세가지 필요조건(항목 1-3)들가운데서 한가지가 발생하지 못하도록 방지하는것이다. 직접적인 교착예방방법은 순환기다림(항목 4)이 발생하지 못하도록 방지하는것이다. 이제 네가지 조건가운데서 매개 조건에 관계되는 수법들을 보기로 한다.

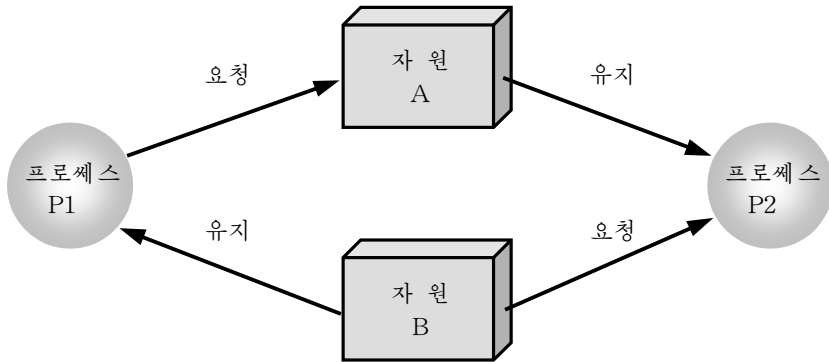



그림 6-5. 순환기다림

호상배제

일반적으로 네가지 조건가운데서 첫번째 조건은 금지할수 없다. 만일 자원접근이 호상배제를 필요로 한다면 호상배제는 조작체계에 의하여 보장되여야 한다. 파일과 같은 일부 자원들은 읽기를 위한 다중접근과 쓰기를 위한 배타적인 접근만을 허용할수 있다. 이러한 경우에도 한개이상의 프로쎄스가 쓰기허용을 요구한다면 교착이 발생할수 있다.

유지 및 기다림

프로쎄스가 필요한 모든 자원들을 한번에 요청하도록 요구하고 모든 요청들이 동시에 허가될수 있을 때까지 프로쎄스를 페색하여 유지기다림조건을 방지할수 있다. 이 방법은 두가지 측면에서 비효과적이다. 우선 어떤 프로쎄스는 모든 자원을 충분하리만큼 요청하도록 긴 시간동안 유지될수 있는데 이때 그 프로쎄스는 일부 자원들과만 진행하였을수 있다. 다른 문제는 어떤 프로쎄스가 자기가 요구하는 모든 자원을 미리 알수 없는것이다.

또한 응용을 위한 모듈식프로그람작성법이나 다중스레드식구조를 사용할 때 발생되는 실제적인 문제가 있다. 응용에서는 동시에 일어나는 요청을 처리하기 위하여 모든 준위 또는 모든 모듈들에서 요청하게 될 모든 자원들을 알아야 할 필요가 제기된다.

비선취권

이 조건은 몇가지 방법으로 방지할수 있다. 우선 만일 어떤 자원들을 유지하고 있는 프로쎄스가 그이상의 요청을 거절당하면 그 프로쎄스는 자기의 본래의 자원을 해방하여야 한다. 그리고 필요하다면 보충적인 자원과 함께 그것을 다시 요청한다. 또한 만일 어떤 프로쎄스가 다른 프로쎄스에 의하여 현재 유지되고 있는 자원을 요청한다면 조작체계는 두번째 프로쎄스를 선취하고 그것이 자원을 해방하도록 요구할수 있다. 이 방법에 의해 두개의 프로쎄스가 동일한 우선권을 가지지 않는 경우에만 교착을 예방할수 있다.

이러한 방법은 자기의 상태를 쉽게 보관하거나 후에 회복할수 있는 자원에 적용할 때에만 실용적이다.

순환기다림

순환기다림조건은 자원형태의 선형적인 순서를 정의함으로써 방지할수 있다. 만일 프로쎄스가 R형태의 자원을 할당 받았다면 순서에서 R다음의 형태의 자원들만을 계속하여 요청할수 있다.

이 방책의 작업을 리해하기 위하여 매개 자원형태에 첨수를 붙인다. 그러면 $i< j$일 때 순서에서 자원R_i는 R_j보다 앞선다. 이제 두 프로쎄스 A와 B가 있을 때 A가 자원 R_i를 차지하고 R_j를 요청하고 B가 R_j를 차지하고 R_i를 요청하는것으로 하여 그것들이 교착되였다고 하자. 이 조건은 $i<j$ 와 $j<i$의 뜻을 다 포함하기때문에 불가능하다.

유지기다림예방과 같이 순환기다림예방은 프로쎄스들을 지연시키고 자원접근을 불필요하게 거부하므로 비효률적이다.

제 3 절. 교착의 회피

교착문제를 해결하는 한가지 방법은 교착예방과는 달리 교착을 피하기하는것이다[2]. **교착예방**에서는 교착의 네가지 조건가운데서 적어도 한가지를 방지하기 위하여 자원요청을 제한한다. 이것은 세가지 필요조건(호상배제, 유지 및 기다림, 비선취권)중의 한가지를 예방하는 방법으로 간접적으로 수행되거나 순환기다림을 예방하는 방법으로 직접적으로 수행된다. 한편 **교착회피**는 세가지 필요조건은 허용되지만 교착점에 확실히 도달하지 못하도록 신중히 선택한다. 그러므로 교착회피는 교착예방보다 높은 병행성을 허용한다. 교착회피를 하는 경우에 현재의 자원할당을 요청하겠는지 안하겠는지 하는 판단은 자동적으로 진행된다. 만일 자원할당요청을 허락한다면 그것은 잠재적으로 교착에로 이끌어간다. 따라서 교착회피는 앞으로 있게 될 프로쎄스자원요청들에 대한 정보를 요구한다.

이 절에서는 교착회피에 대한 두가지 방법을 설명한다.

- 만일 프로쎄스의 요청들이 체계를 교착에로 이끌어 간다면 프로쎄스는 기동하지 않는다.
- 이 할당이 체계를 교착에로 이끌어 간다면 자원요청의 증대를 허용하지 않는다.

프로쎄스시동거부

프로쎄스가 n개이고 자원형태가 m개인 체계를 고찰하자. 다음의 벡토르와 행렬을 정의하자. 즉

자원벡토르 = (R_1, R_2, …, R_n) 체계에 속한 매개 자원의 총수
사용가능벡토르 = (V_1, V_2, …, V_n) 프로쎄스에 할당되지 않은 매개 자원의 총수

요구행렬 = $\begin{pmatrix} C11 & C12 & \cdots & C1n \\ C21 & C22 & \cdots & C2n \\ \vdots & \vdots & \vdots & \vdots \\ C11 & C12 & \cdots & Cnm \end{pmatrix}$ 매개 자원에 대한 매개 프로쎄스의 요구

---

2. 회피라는 용어는 조금 혼돈하기 쉽다. 사실상 이 절에서 취급한 방법들은 교착의 발생을 실제로 방지할수 있기때문에 교착예방의 실례들이라고 볼수 있다.

$$\text{할당행렬} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ A_{11} & A_{12} & \cdots & A_{nm} \end{pmatrix} \quad \text{현재의 할당}$$

요구행렬은 매개 자원에 대한 매개 프로쎄스의 최대요구를 표시하는데 한개의 행은 매개 프로쎄스에 해당한것이다. 즉 C_{ij} = 자원 j에 대한 프로쎄스 i의 요구이다. 이 정보는 교착회피작업에 들어 가기전에 미리 프로쎄스에 의하여 선언되여야 한다. 마찬가지로 A_{ij} = 프로쎄스 i에 대한 자원 j의 현재의 할당이다. 다음의 관계가 성립한다는것을 알수 있다.

1. 모든 i에 대하여 $Ri = V_i + \sum_{k=1}^{n} A_{ki}$ 모든 자원을 사용하거나 배당한다.
2. 모든 k, i 에 대하여 $C_{ki} \leqq R_i$ 프로쎄스는 체계의 총 자원수이상은 요구할수 없다.
3. 모든 k, i에 대하여 $A_{ki} \leqq C_{ki}$ 프로쎄스가 본래 필요에 의해 요구하였던것보다 더 많은 임의의 형태의 자원을 프로쎄스에 할당할수 없다.

이러한 정량적관계에 따라 새로운 프로쎄스의 자원요구들에 의하여 교착을 초래할 가능성이 있다면 그것은 시동하지 못하게 하는 교착회피방법을 규정할수 있다. 모든 i에 대하여

$$R_i \geqq C_{(n+1)i} + \sum_{k=1}^{n} C_{ki}$$

일 때에만 새로운 프로쎄스 P_{n+1}을 시동한다. 즉 모든 현존 프로쎄스의 최대요구에 따라 새로운 프로쎄스의 요구까지 만족되는 경우에만 프로쎄스를 시동한다. 이 방법은 제일 악조건 다시말하여 모든 프로쎄스들이 다 같이 최대요구를 내는 경우를 가정하고 있으므로 최적적인것이 못된다.

자원할당거부

은행가알고리듬[3]이라고 부르는 자원할당거부방법은 [DIJK 65]에서 처음으로 제기하였다. 먼저 상태와 안전한 상태에 대한 개념을 정의하는것으로부터 시작하자. 프로쎄스수와 자원수가 각각 고정된 체계를 고찰하자. 임의의 순간에 프로쎄스는 령 또는 그이상의 자원을 할당 받을수 있다. 체계의 **상태**는 단순히 프로쎄스들에 대한 자원들의 현재의 할당이다. 그러므로 상태는 우에서 정의한 두개의 벡토르(자원벡토르와 사용가능자원벡토르)와 두개의 행렬 즉 요구행렬과 할당행렬로 이루어 진다. **안정한 상태**는 교착으로 끝나지 않는 순차렬이 적어도 한개 존재하는 상태이다(즉 모든 프로쎄스들이 끝까지 실행될수 있다.). **불안정한 상태**는 물론 안정하지 못한 상태이다.

이러한 개념은 다음의 실례에서 설명한다. 그림 6-6 ㄱ에서는 네개의 프로쎄스와 세개의 자원들로 이루어 진 체계의 상태를 보여 주고 있다. 자원들인 R_1, R_2 그리고 R_3 의 총 수는 각각 9, 3 그리고 6단위이다. 현재의 상태에서 할당은 네개의 프로쎄스에 진행되여 자원 2의 1단위, 자원 3의 1단위가 사용가능자원으로 된다. 다음과 같은 질문이 제기된다. 즉 이 상태가 안정한 상태인가? 이 질문에 대답하기 위하여 중간질문을 한다. 즉 4

---

3. 은행업무에서 보면 은행은 빌려 줄수 있는 제한된 축적금과 신용대부한도가 적혀 있는 손님명부를 가지고 있다. 손님은 신용대부한도를 보면서 한번에 일부를 대부 받으려고 할수 있다. 손님이 최대대부금을 대부 받은후에 제때에 상환하리라는 담보는 없다. 은행가는 손님들이 제때에 상환하지 못하여 더는 대부할 자금이 충분하지 못할 위험성이 있게 되면 대부를 거절한다.

개의 프로쎄스가운데서 임의의 프로쎄스가 사용가능한 자원을 가지고 끝까지 실행할수 있는가 ? 다시말하여 임의의 프로쎄스에 대한 현재의 할당과 최대요구사이의 차이를 사용가능한 자원으로 충족시킬수 있는가? 명백히 이것은 P1에 대해서는 불가능하다. 그것은 P1이 R1의 1단위만을 가지고 있는데 R1의 2단위이상, R2의 2단위, R3의 2단위를 요구하고 있기때문이다. 그러나 프로쎄스 P2에 R3의 1단위를 주면 P2는 최대로 요구한 자원을 할당 받게 되여 끝까지 실행할수 있다. 이제 이것이 수행되였다고 하자. P2가 완료되면 그것의 자원들은 사용가능한 자원집결소에 반환할수 있다. 그 결과에 이루어 진 상태를 그림 6 –6 ㄴ에 보여 주고 있다. 이제 나머지 프로쎄스들이 완료될수 있는가 하는 질문이 제기될수 있다. 이 경우에 매개 프로쎄스들은 완료될수 있다. P1을 선택하여 요구되는 자원을 할당하고 P1을 완료한 다음 P1의 자원들을 사용가능한 자원집결소에 반환하였다고 하자. 이때의 상태를 그림 6-6 ㄷ에 보여 주고 있다. 다음에 P3을 완료할수 있다. 이때의 상태를 그림 6-6 ㄹ에 보여 주고 있다. 결국 P4는 완료할수 있다. 이 시점에서 보면 모든 프로쎄스들은 끝까지 실행되였다. 따라서 그림 6-6 ㄱ에서 정의한 상태는 안정한 상태이다.

ㄱ)

요구행렬

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

할당행렬

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

자원벡토르

R1	R2	R3
9	3	6

사용가능자원벡토르

R1	R2	R3
0	1	1

ㄴ)

요구행렬

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

할당행렬

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

사용가능자원벡토르

R1	R2	R3
6	2	3

ㄷ)

요구행렬

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

할당행렬

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

사용가능자원벡토르

R1	R2	R3
7	2	3

ㄹ)

요구행렬

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

할당행렬

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

사용가능자원벡토르

R1	R2	R3
9	3	4

그림 6-6. 안정한 상태의 결정

ㄱ-초기상태, ㄴ- P2가 끝까지 실행, ㄷ- P1이 끝까지 실행, ㄹ-P3이 끝까지 실행

이러한 개념에 의하여 프로쎄스와 자원으로 이루어 진 체계가 항상 안정한 상태에 있다는것을 보증하는 다음의 교착회피방책을 생각할수 있다. 어떤 프로쎄스는 자원모임을 요청할 때 요청을 허락하였다고 보고 체계상태를 해당하게 갱신하고 결과적인 상태가 안정한 상태인가를 결정한다. 만일 그렇다면 요청을 허락하고 아니라면 요청을 안전하게 허락할 때까지 프로쎄스를 폐색한다.

그림 6-7 ㄱ의 행렬에 의하여 정의된 상태를 보자. P_1이 R_1에 한개의 보충단위, R3에 한개의 보충단위를 요청하였다고 하자. 만일 요청이 허락되였다고 가정하면 결과적인 상태는 그림 6-6 ㄱ의것과 같이 된다. 우리는 이 상태가 안정한 상태라는것을 보았다. 따라서 이것은 요청에 안전하게 응한다. 이제 그림 6-7 ㄱ의 상태에로 되돌아가 P1이 R1과 R3에서 각각 한개씩의 보충단위를 요청하였다고 하자. 요청이 허락되였다고 하면 그림 6-7 ㄴ의 상태로 넘어 가겠는가? 이것이 안정한 상태인가? 대답은 《아니》이다. 그것은 매개 프로쎄스가 R1의 적어도 한개의 보충단위를 필요로 하고 사용가능한 자원은 아무것도 없기때문이다. 따라서 교착회피에 기초하여 P1에 의한 요청은 거절하고 P1을 폐색하여야 한다.

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

요구행렬

	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

할당행렬

R1	R2	R3
9	3	6

자원벡토르

R1	R2	R3
1	1	2

사용가능자원벡토르

ㄱ)

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

요구행렬

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

할당행렬

R1	R2	R3
0	1	1

사용가능자원벡토르

ㄴ)

그림 6-7. 불안정한 상태의 결정

ㄱ-초기상태, ㄴ-P1이 R1과 R3에서 각각 1단위를 요청

그림 6-7ㄴ의 상태가 교착상태가 아니라는것을 념두에 두어야 한다. 그것은 다만교착의 가능성을 가진다. 실례로 만일 P1이 이 상태에서 실행되였다면 그것은 계속하여 자원들을 다시 필요로 하기전에 R1의 한개 단위와 R3의 한개 단위를 해방할수 있다. 만일 그렇게 된다면 체계는 안정한 상태로 돌아 온다. 결국 교착회피방책은 교착을 확신성 있게 예견하지 못하고 그저 교착의 가능성을 예상하며 그러한 가능성이 결코 없다는것을 담보해 줄뿐이다.

```
struct state
{
    int  resource[m];
    int  available[m];
    int  claim[n][m];
    int  alloc[n][m];
}
```

ㄱ)

```
if  (alloc [i.*] + request [*] > chaim [I, *])
{
    <error>;          /*  -- 총 요청 > chaim*/
}
else  if  (request [*] >  available [*])
{
    < suspend process>;
}
else                    /* -- alloc 모의  */
{
    < define newstate by:
      alloc [I, *]= alloc [I, *] + request [*];
      available [*] = available [*] - request [*] >;
}
if (safe (newstate))
{
    <carry out allocation>;
}
else
{
    <restore original state>;
    <suspend process>;
}
```

ㄴ)

```
boolean safe (state s)
{
    int               currentavail [m];
    process rest [<number of process>];
    currentavail = available;
    possible = true;
    while (possoble)
    {
         find  a  Pk in  rest  such  that
              claim [k, *] -  alloc [k, *]  <=  currentavail;
         If (found) then            /* p의 집행모의 */
         {
            curentavail = currentavail + alloc [k, *];
            rest = rest - {Pk};
         }
         else
             possible = false;
         }
    return (rest == null);
}
```

ㄷ)

그림 6-8. 교착회피론리

ㄱ-전역자료구조, ㄴ-자료할당알고리듬, ㄷ-안정성알고리듬(은행업무알고리듬)의 시험

그림 6-8에서는 교착회피론리를 요약한 내용을 주고 있다. 체계의 기본알고리듬은 ㄴ부분에 있다. 체계의 상태를 자료구조 *state*에 의해 정의하면 *reques t*[*]은 프로쎄스 *i*에 의해 요청된 자원들을 정의하는 벡토르이다. 우선 요청이 프로쎄스의 본래의 요구를 초과하지 않는가를 확인하기 위하여 시험을 진행한다. 만일 요청이 유효이면 다음 걸음에서 요청을 리행하는것이 가능한가(즉 사용가능한 자원이 충분히 있는가)를 결정한다. 만일 가능하지 않으면 프로쎄스는 중단된다. 그러나 만일 가능하면 마지막단계에서 요청을 안전하게 리행할수 있는가를 결정한다. 이것을 하기 위하여 *newstate*를 형성하도록 프로쎄스 *i*에 자원들을 림시로 할당한다. 그러면 안정성시험은 그림 6-8 ㄷ의 알고리듬을 사용하여 할수 있다.

교착회피는 교착검출에서와 같이 프로쎄스들을 선취하거나 재연산할 필요가 없고 교착예방에서보다 제한이 적다는 우점을 가지고 있다. 그러나 그 사용에서는 제한조건이 많다. 즉

- 매개 프로쎄스에 대한 최대자원요구는 미리 서술해 주어야 한다.
- 고찰중의 프로쎄스들은 독립적이여야 한다. 즉 그것들이 진행되는 순서는 임의의 동기화요구에 의하여 제한을 받지 말아야 한다.
- 할당하여야 할 자원수는 고정되여 있어야 한다.
- 자원을 유지하는 동안에는 어떤 프로쎄스도 탈퇴하지 말아야 한다.

제 4 절. 교착의 검출

교착예방방책은 매우 보수적인 방법이다. 그것은 자원접근을 제한하고 프로쎄스들에 제한조건을 주는 방법으로 교착문제를 해결하고 있다. 정반대로 교착검출방책에서는 자원접근을 제한하거나 프로쎄스작업을 구속하지도 않는다. 교착검출을 하면 요청되는 자원들을 필요한 때에는 언제나 프로쎄스에 보장한다. 조작체계는 우에서 지적한 조건 (4)와 그림 6-5에서 설명한 순환기다림조건을 검출하도록 하는 알고리듬을 주기적으로 집행한다.

교착검출알고리듬

교착의 검사는 자주는 매개 자원요청 때마다 드문히는 교착이 발생할가 하는 시점에서 진행한다. 매개 자원요청 때마다 진행하는 검사방법은 두가지 우점을 가지고 있다. 즉 교착을 초기에 검출할수 있고 그것이 체계의 상태의 증분변화에 기초하고 있으므로 알고리듬이 상대적으로 단순한것이다. 다른 한편 그러한 빈번한 검사로 인하여 처리기의 시간이 현저히 소비된다.

교착검출의 공통알고리듬을 [COFF 71]에서 설명하고 있다. 이 알고리듬에서는 앞절에서 설명한 할당행렬과 사용가능자원벡토르를 사용한다. 또한 요청행렬 **Q**는 프로쎄스 *i*에서 요청되는 형태 *j*의 자원수를 *qij*로 표현하는것과 같이 정의한다. 알고리듬은 시행되면서 교착되지 않는 프로쎄스들에 표식을 달아 준다. 이때 다음과 같은 걸음들이 실행된다. 즉

1. 할당행렬에서 모두가 령인 행을 가지는 매개 프로쎄스들을 표식한다.
2. 사용가능자원벡토르나 동일한 림시벡토르 **W**를 초기화한다.
3. 프로쎄스 *i*가 현재 표식달지 않고 **Q**의 *i*번째 행이 **W**보다 작거나 같은 경우의 첨수 *i*를 찾는다. 즉 $1 \leq k \leq m$에 대하여 $Q_{ik} \leq W_k$이다. 만일 그러한 행이 발견되지 않으면 알고리듬을 끝낸다.
4. 만일 그러한 행이 발견되면 프로쎄스 *i*에 표식을 달고 할당행렬의 해당한 행을

W에 더한다. 즉 $1 \leqq k \leqq m$에 대하여 $W_k = W_k + A_{ik}$를 설정한다. 그리고 걸음 3에로 간다.

만일 알고리듬의 끝에 표식달지 않은 프로쎄스들이 있다면 교착이 존재한다. 매개 표식달지 않은 프로쎄스는 교착된다. 이 알고리듬에서 수법은 자기의 자원요청들을 사용가능한 자원들로 충족시킬수 있는 프로쎄스를 찾아 내는것이다. 이때 그 자원들은 허가되고 프로쎄스는 끝까지 실행되며 그것의 모든 자원들은 해방된다고 가정한다. 다음에 알고리듬은 만족되는 다른 프로쎄스를 찾는다. 지적할것은 이 알고리듬이 교착예방을 담보하지 못한다는것이다. 알고리듬은 요청들이 허가되는 순서에 관계된다. 만일 교착이 현재 존재한다고 하면 알고리듬은 하던 모든것을 끝낸다.

교착검출알고리듬을 설명하는데 그림 6-9를 사용할수 있다. 알고리듬은 다음과 같이 실행된다.

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

요청행렬

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

할당행렬

R1	R2	R3	R4	R5
2	1	1	2	1

자원벡토르

R1	R2	R3	R4	R5
0	0	0	0	1

사용가능자원벡토르

그림 6-9. 교착검출의 실례

1. P4가 할당된 자원을 가지고 있지 않으므로 P4에 표식을 단다.
2. W = (0 0 0 0 1)으로 설정한다.
3. 프로쎄스 P3의 요청은 W보다 작거나 같으므로 P3에 표식을 달고
   W = W + (0 0 0 1 0) = (0 0 0 1 1)으로 설정한다.
4. 표식을 달지 않는 다른 프로쎄스는 W보다 작거나 같은 행을 Q에 가지고 있지 않다. 그러므로 알고리듬을 끝낸다.

알고리듬은 P1과 P2에 표식을 달지 않고 끝을 내는데 그것은 이 프로쎄스들이 교착된다는것을 표시한다.

회복

일단 교착이 검출되였다면 그것을 회복하기 위한 어떤 방책이 필요하다. 가능한 방법들을 복잡성이 증가되는 순서로 목록화하여 아래에 제시한다. 즉

1. 교착된 모든 프로쎄스들을 포기한다. 이것이 조작체계에서 적용하는 가장 일반적인 해결방책의 하나이다.
2. 교착된 매개 프로쎄스를 미리 정의한 검사점에로 후퇴시킨 다음 모든 프로쎄스들을 재시동한다. 이것은 재연산 및 재시동기구가 체계안에 구축되여 있을것을 요구한다. 이 방법의 위험성은 본래의 교착이 재발생될수 있는것이다. 그러나 병행처리의 비확정성은 보통 이것이 일어 나지 않는다는것을 담보한다.
3. 교착이 더는 존재하지 않을 때까지 교착된 프로쎄스들을 련속 포기한다. 프로쎄

스를 포기시키기 위해 선택되는 순서는 어떤 최소간격기준에 의거하여야 한다. 매개 포기후에 아직도 교착이 존재하는가를 알아 보기 위하여 검출알고리듬을 다시 호출한다.

4. 교착이 더는 존재하지 않을 때까지 자원들을 련속 선취한다. (3)에서와 같이 가격에 준한 선택을 사용해야 한다. 매개 선취후에 검출알고리듬의 재호출이 필요하다. 그것으로부터 선취된 자원을 가지는 프로쎄스는 그 자원이 수집되기전의 실행점에로 재연산되여야 한다.

(3)과 (4)의 선택기준은 다음의것들중에서 한가지가 될수 있다. 다음의 프로쎄스를 선택한다.

- 이제까지 소비한 처리시간이 제일 적은 프로쎄스
- 이제까지 진행한 출력이 제일 적은 프로쎄스
- 대략적인 나머지 시간이 제일 큰 프로쎄스
- 지금까지 할당한 전체 자원이 제일 적은 프로쎄스
- 우선권준위가 제일 낮은 프로쎄스

이 량들중의 일부는 다른것들보다 더 쉽게 측정할수 있다. 대략적인 나머지 시간이 특별히 주목된다. 또한 우선권의 측정수단이외에는 체계 전체로서의 가격에 대비되는 사용자의 《가격》에 대한 지적은 되여 있지 않다.

제 5 절. 통합적교착해결방책

표 6-1에서 지적한바와 같이 교착을 취급하는 모든 방책에는 우단점들이 있다. 이러한 방책들가운데서 한가지만을 적용하는 조작체계를 설계하려고 하기보다 각이한 정황속에서 각이한 방책을 사용하는것이 더 효과적일것이다. [SILB 98]에 한가지 방도를 제시하였다. 즉

- 자원들을 서로 다른 여러가지 부류로 그룹을 묶는것
- 자원부류들사이에서의 교착을 예방할수 있도록 순환고리를 막기 위하여 이미 정의한 선형순서화방법을 사용하는것
- 자원부류안에서는 그 부류에 제일 적합한 알고리듬을 사용하는것

이 수법의 실례로서 다음의 자원부류를 고찰하자

- **교체가능공간:** 교체프로쎄스에서 사용하기 위한 2차기억기의 기억블로크
- **프로쎄스자원:** 테프구동기와 같은 지정가능한 장치와 파일들
- **주기억기:** 페지나 토막들을 프로쎄스에 할당할수 있는것
- **내부자원:** 입출력통로와 같은것

진행표의 순서는 자원이 할당되는 순서를 표시한다. 프로쎄스가 그것의 수명기간 따를수 있는 걸음들의 순차를 고려하면 이 순서가 합리적인것이다. 매개 부류안에서는 다음과 같은 방책을 사용할수 있다.

- **교체가능공간:** 유지기다림예방방책에서와 같이 사용가능한 필요한 모든 자원을 한번에 할당하도록 요구함으로써 교착을 예방한다. 이 방책은 최대기억기요구사항을 알고 있는 경우에 흔히 사용할수 있는 합리적인 방법이다. 역시 교착회피가 가능하다.

- **프로쎄스자원:** 이러한 부류에서는 교착회피가 효과적이다. 그것은 이 부류에서 요구하는 자원들을 프로쎄스가 시간적으로 앞서 선언하도록 하는것이 합리적이기때문이다. 이 부류안에서는 순서짓기하는 자원에 의하여 교착예방이 역시 가능하다.
- **주기억기:** 선취에 의한 교착예방은 주기억기에 대하여 가장 적합한 방책이라고 할수 있다. 프로쎄스가 선취되면 그것은 교착을 해결하기 위하여 내놓은 자유공간인 2차기억기와 단순히 교체된다.
- **내부자원:** 자원의 순서짓기에 의하여 교착예방에 사용할수 있다.

제 6 절. 철학자식사문제

어떤 때 어느 곳에 다섯명의 철학자가 같이 살고 있었다. 수년간에 걸치는 그들의 생활은 기본적으로 사고하고 먹는것이였다. 그들모두는 자기들의 사고를 할수 있게 해주는 유일한 량식은 스퍼게티(속이 비지 않은 마카로니)라고 생각하였다.

식사배치는 단순하였다(그림 6-10). 원탁우에는 스퍼게티를 담은 한개의 큰 사발, 한사람에게 한개씩 차례지게 다섯개의 접시와 다섯개의 포크가 놓여 있었다. 식사하려는 철학자는 탁의 자기 자리에 다가가 량쪽에 놓여 있는 두개의 포크를 사용하여 스퍼게티를 집어 먹는다. 이 문제는 철학자들이 식사하는 례식(알고리듬)을 작성하는것이다. 알고리듬은 호상배제(두명의 철학자는 동일한 순간에 동일한 포크를 사용할수 없다.)조건을 만족하여야 하며 교착과 고갈(이 경우에 항목은 문자 또는 알고리듬적인 의미를 가진다.)을 피하기해야 한다.

딕스트라에 의해 제기된 이 문제는 그 자체로서는 중요하다거나 그 어떤 련관이 있는 문제로 보이지 않을수 있다. 그러나 이것은 교착이나 고갈에서 제기되는 기본문제를 설명하여 주고 있다. 더우기 해답을 찾는 과정에는 병행프로그람작성과 관련되는 많은 어려운 점들이 제기된다 ([GING 90]을 볼것). 더우기 철학자식사문제는 공유기억기의 정합을 취급하는 문제의 전형이라고 볼수 있는데 이러한 문제는 프로그람에서 병행스레드의 집행을 포함하는 경우에 발생할수 있다. 따라서 이 문제는 동기화에 대한 방법을 평가하는데서 표준적인 시험문제로 된다.

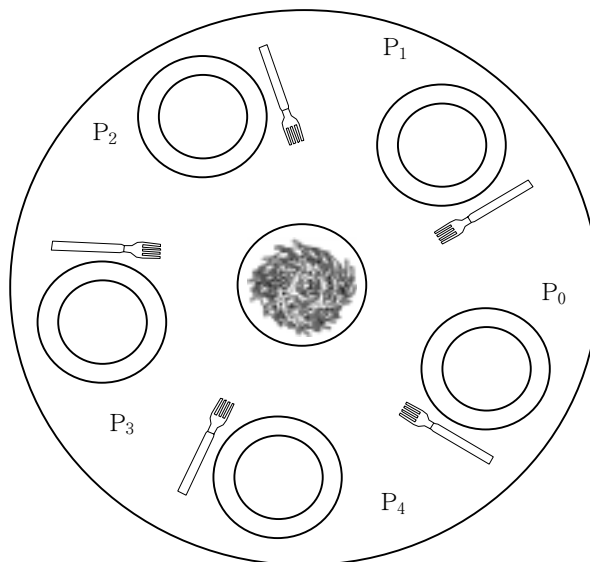



그림 6-10. 철학자들의 식사배치

그림 6-11에는 신호기를 사용한 해결방도를 제기하고 있다. 매개 철학자는 먼저 왼쪽에 있는 포크를 쥐고 다음에 오른쪽에 있는 포크를 쥔다. 철학자가 식사를 끝낸 다음에 두개의 포크를 탁에 다시 놓는다. 이 해결방도는 교착에로 이끌어 간다. 만일 모든 철학자들이 같은 시각에 배가 고파서 그들모두가 자리에 앉아서 자기 왼쪽에 있는 포크를 쥐고 다른 포크를 잡으려 손을 내 민다. 그런데 거기에는 포크가 없다. 이러한 자세에서 모든 철학자들은 단식을 한다.

```
/* 철학자식사프로그람 */
semaphore  fork [5] = { 1 };
int  i;
void  philosopher ( int  i)
{
      while  (true)
      {
            think();
            wait (fork [i]);
            wait (fork  [(i+1)  mod 5]);
            eat();
            signal(fork [(i+1) mod 5]);
            signal(fork[i]);
       }
}
void main()
{
       parbegin (philosopher (0), philosopher (1), philosopher (2),
           philosopher (3), philosopher (4));
}
```

그림 6-11. 철학자식사문제의 첫번째 해결방도

교착의 위험성을 극복하기 위하여 다섯개의 포크를 더 사다 주든가(보다 위생적인 해결방도!) 철학자들에게 한개의 포크로 스퍼게티를 먹으라고 할수 있을것이다. 다른 해결방도로서 네명의 철학자만이 식당에 들어 오도록 하는 경우를 생각할수 있다. 기껏해서 네명의 철학자들이 앉으면 적어도 한명의 철학자는 두개의 포크를 쥘수 있게 된다. 그림 6-12에는 역시 신호기를 사용한 해결방도를 제시하였다. 이 해결방도에서는 교착과 고갈을 발생시키지 않는다.

```
program  dinigphilosophers;
semaphore  fork[5] = {1};
semaphore  room = {4};
int  i;
void  philosopher  (int i);
{
    while  (true)
         think ();
```

```
        wait  (room);
        wait  (fork[i]);
        wait  (fork [ (i+1)  mod  5]);
        eat();
        signal  (fork [(i+1)  mod 5]);
        signal  (fork[i]);
        signal  (room);
}

void  main()
{
      parbegin  (philosopher (0), philosopher (1), philosopher (2),
                 philosopher (3), philosopher (4)
}
```

그림 6-12. 철학자식사문제의 두번째 해결방도

제 7 절. UNIX의 병행기구

UNIX에서는 프로쎄스사이의 통신과 동기화를 보장하기 위한 여러가지 기구들을 제공하고 있다. 여기에서는 제일 중요한것을 보기로 한다. 즉

- 흐름관
- 통보문
- 공유기억기
- 신호기
- 신호

흐름관들, 통보문들 및 공유기억기는 프로쎄스들을 거쳐 자료를 통신하는 수단을 주며 신호기들과 신호들은 다른 프로쎄스들에 의하여 작업을 시동시키는데 사용된다.

흐름관

조작체계의 발전에 UNIX가 기여한 가장 큰 공적들중의 하나가 바로 흐름관이다. 협동루틴의 개념[RITC 84]에 의하여 흐름관은 두 프로쎄스가 생산자-소비자모형우에서 통신할수 있게 하는 순환완충기이다. 이와 같이 그것은 선입선출대기렬로서 한개의 프로쎄스에 대해서는 쓰기를 진행하고 다른 프로쎄스에 의해서는 읽기를 진행한다.

흐름관이 창조되면 그것에는 고정된 바이트단위의 크기가 주어 진다. 프로쎄스가 흐름관에 쓰기를 진행할 때 충분한 공간이 있으면 쓰기요청이 즉시에 진행되고 그렇지 않으면 프로쎄스는 페색된다. 마찬가지로 흐름관에서 읽기를 진행할 때 흐름관에 현재 있는것보다 더 많은 바이트를 읽으려고 한다면 읽기프로쎄스는 페색되고 그렇지 않으면 읽기요청이 즉시에 집행된다. 조작체계는 호상배제를 실시한다. 즉 한개의 프로쎄스만이 흐름관을 접근할수 있게 한다.

흐름관에는 두가지 형태 즉 이름 붙인것과 이름 없는것이 있다. 관련되는 프로쎄스들만이 이름없는 흐름관들을 공유할수 있고 관련되지 않는 프로쎄스들은 이름 붙인 흐름관들을 공유할수 있다.

통보문

통보문은 동봉하는 형태를 가진 본문의 불로크이다. UNIX에서는 통보문넘기기를 수행하는 프로쎄스들을 호출하는 *msgnd* 및 *msgrcv*체계를 두고 있다. 매개 프로쎄스가 관계하는것은 우편통과 같은 기능을 수행하는 통보문대기렬이다.

통보문송신자는 보내는 통보문과 함께 통보문의 형태를 명시해 준다. 그러면 이것은 수신자의 선택기준으로 사용할수 있다. 수신자는 선입선출순서로 통보문을 회복하거나 형태별로 회복한다. 충만된 대기렬에로 통보문을 송신하려고 할 때에는 프로쎄스가 중단된다. 또한 빈 대기렬로부터 읽으려고 할 때에도 프로쎄스가 중단된다. 만일 프로쎄스가 어떤 형태의 통보문을 읽으려고 하였는데 그러한 형태의 통보문이 존재하지 않기때문이 실패하였다면 프로쎄스는 중단되지 않는다.

공유기억기

UNIX에서 제공하는 프로쎄스간 통신의 가장 속도가 높은 형태는 공유기억기이다. 이것은 여러개의 처리기들에 의하여 공유되는 가상기억기의 공통블로크이다. 프로쎄스들은 가상기억공간의 다른 구역을 읽고 쓰는데 사용하는 동일한 기계명령들을 사용하여 공유기억기를 읽고 쓴다. 프로쎄스에 따라 읽기전용이나 읽고쓰기가 허용되는데 그것은 프로쎄스당 기준에 의해 결정된다. 호상배제제약조건들은 공유기억기기능의 일부가 아니고 공유기억기를 사용하고 있는 프로쎄스에 의하여 주어 져야 한다.

신호기

UNIX System V에서 호출하는 신호기체계는 제5장에서 정의한 *wait*와 *signal*기본지령들을 일반화한것이다. 그것에서는 몇가지 연산들을 동시에 수행할수 있으며 증가 및 감소 연산들은 1보다 더 큰 값일수 있다. 핵심부는 요청된 모든 연산들을 자동적으로 수행한다. 다른 프로쎄스들은 모든 연산이 수행될 때까지 신호기를 호출할수 없다.

신호기는 다음의 요소들로 구성된다.

- 신호기의 현재값
- 신호기에 의하여 연산하기 위한 마지막프로쎄스의 ID
- 신호기의 값이 현재의 값보다 더 커지기를 기다리고 있는 프로쎄스의 수
- 신호기의 값이 령이 되기를 기다리는 프로쎄스의 수

신호기와 관련되는것은 그 신호기에 의해 중단된 프로쎄스들의 대기렬이다.

신호기들은 실제적으로 모임형태로 창조되는데 신호기모임은 한개 또는 그이상의 신호기들로 이루어 진다. 모임안의 모든 신호기값들이 동일한 순간에 설정되게 하는 *semctl*체계호출이 있다. 또한 신호기모임의 매개 신호기마다에서 정의되는 신호기연산들의 목록을 한개의 인수로 취하는 *semop*체계호출이 있다. 이것이 호출될 때 핵심부는 지시된 연산들을 한번에 수행한다. 매개 연산에 대하여 실제기능은 값 *sem_op*에 의하여 명시된다. 이때 다음과 같은 가능성들이 있다.

- 만일 *sem_op*가 정이면 핵심부는 신호기의 값을 1증가시키고 신호기의 값이 증가되기를 기다리는 모든 프로쎄스들을 깨운다.
- 만일 *sem_op*가 0이면 핵심부는 신호기의 값을 검사한다. 만일 0이면 그 목록의 다른 연산들을 계속한다. 그렇지 않으면 이 신호기가 0이 되기를 기다리는 프로쎄스를 1증가시키고 신호기의 값이 0과 같은 사건에 의거하는 프로쎄스는 중단한다.

- *Sem_op*가 부이고 그것의 절대값이 신호기의 값보다 작거나 같으면 핵심부는 신호기의 값에 *sem_op*(부의 수)를 더한다. 만일 결과가 0이면 핵심부는 신호기의 값이 0과 같아 지기를 기다리는 모든 프로쎄스들을 깨운다.
- 만일 *sem_op*가 부이고 그것의 절대값이 신호기의 값보다 더 크면 핵심부는 신호기의 값을 증가시키는 사건에 의거하는 프로쎄스를 중단한다.

신호기의 이러한 일반화는 프로쎄스의 동기화와 정합을 수행하는데서 현저한 유연성을 보장한다.

신호

신호는 비동기적인 사건의 발생을 프로쎄스에게 통지하는 쏘프트웨어적인 기구이다. 신호는 장치적인 새치기와 비슷하지만 우선권은 사용하지 않는다. 즉 모든 신호들은 동일하게 취급된다. 동일한 순간에 발생하는 신호들은 특별한 순서짓기가 없이 한번에 프로쎄스에 제공된다.

프로쎄스들은 신호들을 서로 송신할수 있다. 핵심부는 신호들을 내부적으로 송신할수 있다. 신호는 송신되는 신호를 받고 있는 프로쎄스를 위한 프로쎄스표의 어떤 마당을 갱신하는 방법으로 배포된다. 매개 신호는 한개의 비트로서 유지되므로 주어 진 형태의 신호들은 대기렬을 지울수 없다. 신호는 실행을 위해 프로쎄스를 깨우거나 프로쎄스가 체계호출로부터 복귀하기 위해 준비되거나 하면 즉시 처리된다. 프로쎄스는 신호처리기능을 지향하는 어떤 지정된 작업(실례로 완료)을 수행함으로써 신호에 응답할수 있다.

표 6-2에는 UNIX SVR 4에서 정의하고 있는 신호의 목록을 제시하였다.

제 8 절. Solaris의 스레드동기화기본지령

UNIX SVR 4의 병행기구외에 Solaris는 네가지의 스레드동기화기본지령을 지원하고 있다.

- 호상배제(mutex)의 페쇄
- 신호기
- 다중읽기자, 단일쓰기자(읽기자/쓰기자)의 페쇄
- 조건변수

Solaris에서는 핵심부스레드를 위한 핵심부안에서 이 기본지령들을 실현하고 있다. 또한 사용자준위의 스레드용스레드서고에도 이 기본지령들을 제공하고 있다. 기본지령이 집행되면 창조되는 스레드에 의하여 명시되는 파라메터들은 포함하는 자료구조가 창조된다(그림 6-13). 일단 동기화객체가 창조되면 넣기(차지, 페쇄)와 해방(페쇄해제)을 수행할수 있는 두가지 연산만이 존재한다. 핵심부나 스레드서고안에는 호상배제를 집행하고 교착을 예방하기 위한 기구가 없다. 만일 스레드가 보호하려고 하는 자료나 코드의 일부를 접근하려고 하면서도 적합한 동기화기본지령을 사용하지 않는다면 그러한 접근이 발생한다. 만일 스레드가 어떤 객체를 페쇄하였다가 페쇄해제가 잘못되였다면 핵심부작업은 진행되지 않는다.

모든 동기화기본지령들은 객체를 시험하는 하드웨어명령이 존재할것을 요구하며 제5장 제3절에서 설명한바와 같이 한개의 자동연산으로 설정된다.

표 6-2. UNIX의 신호

값	이 름	설 명
01	SIGHUP	장치 정지; 프로쎄스의 사용자가 유효작업을 하고 있지 않다는것을 핵심부가 판단하였을 때 프로쎄스에게 보낸다.
02	SINGINT	새치기
03	SIGQUIT	정지; 프로쎄스를 정지시키고 기억기쏟기를 하기 위하여 사용자가 보낸다.
04	SIGILL	비법명령
05	SIGTRAP	추적함정; 프로쎄스추적을 위한 코드집행을 개시한다.
06	SIGIOT	IOT명령
07	SIGEMT	EMT명령
08	SIGFPT	류동소수점제외
09	SIGKILL	소멸; 프로쎄스를 완료한다.
10	SIGBUS	모선오유
11	SIGSEGV	토막화위반; 프로쎄스가 가상기억공간밖의 주소를 접근하려고 한다.
12	SIGSYS	체계호출이 불량인 인수
13	SIGPIPE	자기에게 접속한 읽기자를 가지지 않은 흐름관에 쓰기를 한다.
14	SIGALARM	경보박자; 프로쎄스가 주기시간후에 신호를 받으려고 할 때 낸다.
15	SIGTERM	쏘프트웨어적인 완료
16	SIGUSR1	사용자정의의 신호1
17	SIGUSR2	사용자정의의 신호2
18	SIGCLD	자식의 소멸
19	SIGPWR	전원고장

호상배제의 폐쇄

호상배제의 폐쇄는 폐쇄가 되였을 때 한개이상의 스레드가 진행되는것을 막는다. 호상배제를 폐쇄한 스레드는 그것을 해제할수 있어야 한다. 스레드는 mutex-enter기본지령을 집행함으로써 호상배제의 폐쇄를 하려고 한다. 만일 mutex-enter가 폐쇄를 설정할수 없다면(다른 스레드에 의하여 지어 그것이 설정되여 있기때문에) 폐색작업은 호상배제객체에 보관된 형태명시정보에 의거해야 한다. 기정폐색방법은 회전폐쇄이다. 폐색된 스레드는 폐쇄상태를 등록하고 그동안 회전기다림고리안에서 집행을 계속한다. 새치기에 기초한 폐색기구는 선택적인 기구이다. 후자의 경우에 호상배제는 이 폐쇄에 의해 잠자고 있는 스레드의 대기렬을 식별하는 *turnstile id*를 포함한다.

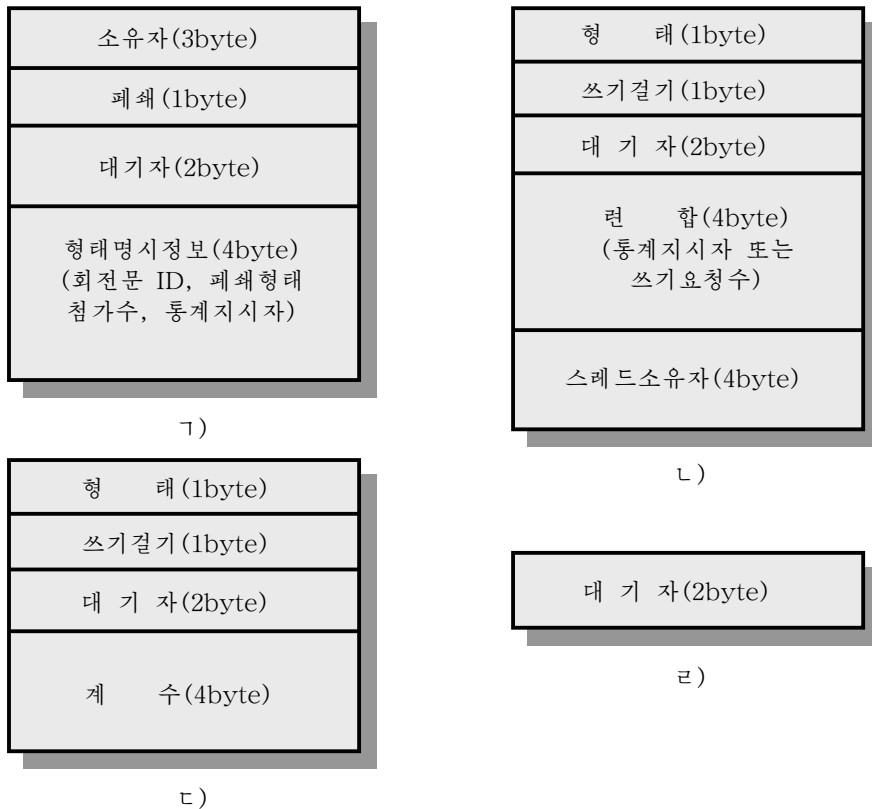



그림 6-13. Solaris의 동기화자료구조
ㄱ-호상배제의 페쇄, ㄴ-신호기, ㄷ-읽기자/쓰기자의 페쇄, ㄹ-조건변수

호상배제의 페쇄와 관련된 기본지령들은 다음과 같다.

mutex_enter() 페쇄를 하여 그것이 이미 유지되고 있다면 잠재적으로 페색한다.
mutex_exit() 페쇄를 해방하며 잠재적으로 대기자의 페색을 푼다.
mutex_tryenter() 페쇄가 이미 유지되고 있지 않으면 페쇄를 한다.

mutex_tryenter()기본지령은 호상배제기능을 수행하는 비페색방법을 주고 있다. 이것은 프로그람작성자로 하여금 사용자수준의 스레드를 위한 바쁜기다림방법을 사용할수 있게 하는데 이것은 한개의 스레드가 페색되므로 옹근 프로쎄스가 페색되는것을 피할수 있게 한다.

신호기

Solaris는 다음의 기본지령을 가지고 전형적인 계수형신호기들을 제공하고 있다. 즉

sema_p() 신호기를 1감소시켜 스레드를 잠재적으로 페색한다.
sema_v() 신호기를 1증가시켜 기다리고 있는 스레드의 페색을 잠재적으로 푼다.
sema_tryp() 페색의 요구가 없으면 신호기를 1감소시킨다.

특히 sema_tryp()기본지령은 바쁜기다림을 허용한다.

읽기자/쓰기자의 폐쇄

읽기자/쓰기자의 폐쇄는 다중스레드로 하여금 폐쇄에 의하여 조절되고 있는 객체에 동시적인 읽기전용의 접근을 할수 있게 한다. 그것은 또한 단일스레드가 모든 읽기자들을 배제하고 있는 동안 어떤 순간에 쓰기를 하여 객체에 접근할수 있게 한다. 쓰기를 위한 폐쇄를 하였을 때 그것은 *write lock*의 상태를 취한다. 읽기나 쓰기를 위해 접근하려는 모든 스레드들은 기다려야 한다. 만일 한개 또는 그이상의 읽기자들이 폐쇄를 당했다고 하면 그것의 상태는 *read lock*이다. 기본지령들은 다음과 같다.

rw_enter()	읽기자 또는 쓰기자로서의 폐쇄를 하려고 시도한다.
rw_exit()	읽기자 또는 쓰기자로서 폐쇄를 해제한다.
rw_tryenter()	폐색이 요구되지 않으면 폐쇄를 한다.
rw_downgrade()	쓰기폐쇄를 한 스레드는 그것을 읽기폐쇄로 바꾼다. 기다리고 있는 임의의 쓰기자는 이 스레드가 폐쇄를 해제할 때까지 그냥 기다린다. 만일 기다리고 있는 쓰기자가 없으면 기본지령은 임의의 미정의 읽기자들을 깨운다.
rw_tryupgrade()	읽기자의 폐쇄를 쓰기자의 폐쇄로 바꾸려고 한다.

조건변수

조건변수는 특정한 조건이 진실로 될 때까지 기다리는데 사용된다. 조건변수는 호상배제와 협력하여 사용하여야 한다. 이것은 그림 5-22에 제시한 형태의 감시기를 실현한다. 기본지령들은 다음과 같다.

cv_wait()	조건이 신호를 할 때까지 패색한다.
cv_signal()	cv_wait()로 페색된 스레드한개를 깨운다.
cv_broadcast()	cv_wait()로 페색된 스레드모두를 깨운다.

cv_wait()는 페색하기전에 관련된 호상배제를 해제하고 복귀하기전에 그것을 재획득한다. 호상배제의 재획득은 호상배제를 기다리고 있는 다른 스레드에 의하여 페색될수 있기때문에 기다림을 발생시킨 조건은 재시동되여야 한다. 대표적인 사용법은 다음과 같다.

```
mutex_enter(&m)
        ⋮
while (some_condition){
  cv_wait(&cv, &m);
}
        ⋮
mutex_exit(&m);
```

이것은 조건이 복잡한 식으로 되게 한다. 그것은 호상배제에 의하여 보호되기때문이다.

제 9 절. Windows 2000의 병행기구

Windows 2000(W2K)에서는 객체구성방식의 일부로서 스레드들사이에 동기화를 보장하고 있다. 동기화기능을 실현하기 위하여 W2K가 사용하는 기구는 동기화객체들의 계렬인데 그것은 다음의것들로 이루어 져 있다.

표 6-3. Windows 2000의 동기화객체

객체형태	정의	신호를 내는 상태로 설정하는 시간	기다리고 있는 스레드에 대한 효과
프로쎄스	프로그람을 실행하는데 필요한 주소공간과 자원들을 포함하는 프로그람기동	마지막스레드를 끝냈을 때	모두 해방
스레드	프로쎄스내에서 집행될수 있는 실체	스레드를 끝냈을 때	모두 해방
파일	열린 파일이나 입출력장치의 구체례	입출력연산을 끝냈을 때	모두 해방
조종탁입력	본문창문화면완충기 (구체례로 MS-DOS응용에서 화면입출력을 처리하는데 사용)	입력을 처리에 리용할수 있을 때	한개의 스레드를 해방
파일변경통지	임의의 파일체계의 변화를 알리는 통지서	이 객체의 선택기준과 일치하는 파일체계에서 변화가 발생하였을 때	한개의 스레드를 해방
호상배제	Win 32와 OS/2 환경에서 호상배제기능을 제공하는 기구	소유하고 있는 스레드나 다른 스레드가 돌연변이를 해제하였을 때	한개의 스레드를 해방
신호기	자원을 사용할수 있는 스레드의 수를 조절하는 계수기	신호기계수가 0으로 될 때	모두 해방
사건	체계의 사건이 발생하였다는것을 알리는 통고	스레드가 사건을 설정하였을 때	모두 해방
기다림시계	시간의 경과를 기록하는 계수기	설정시간이 되였거나 시간간격이 만기되였을 때	모두 해방

주의 : 어두운 부분은 동기화의 유일한 목적을 위해 존재하는 객체에 해당한다.

- 프로쎄스
- 스레드
- 파일
- 조종탁입력
- 파일변경통지
- 호상배제
- 신호기
- 사건
- 기다림시계

우에서 지적한 목록에서 첫 네가지의 객체형태는 다른 목적에 사용되는것이긴 하지만 동기화를 위해서도 역시 사용할수 있다. 나머지 객체형태들은 동기화를 지원하기 위하여 특별히 설계된것이다.

매개 동기화객체의 구체례는 신호를 내거나 또는 내지 않는 상태에 있을수 있다. 스레드는 신호를 내지 않는 상태에 있는 객체에서 중단될수 있다. 스레드는 객체가 신호를 내는 상태에 들어 갈 때 해방된다. 기구는 간단하다. 스레드는 동기화객체의 례외처리를 사용하여 W2K집행부에 기다림요청을 낸다. 객체가 신호를 내는 상태에 들어 갈 때 W2K집행부는 그 동기화객체를 기다리고 있는 모든 스레드의 객체들을 해방한다.

표 6-3에서는 매개 객체형태가 신호를 내는 사건들과 그것이 기다리고 있는 스레드에 미치는 효과를 종합하였다.

호상배제의 객체는 호상 배타적인 자원접근에 사용하는데 이때 한번에 오직 한개의 스레드객체만이 접근을 하도록 한다. 따라서 그것은 오직 신호기로서의 기능을 수행한다. 호상배제의 객체가 신호를 내는 상태에 들어 갈 때 호상배제하에서 기다리는 오직 한개의 스레드만이 해방된다. 호상배제들은 각이한 프로쎄스들에서 실행되고 있는 스레드들을 동기화하는데 사용할수 있다.

호상배제들과 같이 신호기들은 다중프로쎄스들에서의 스레드들에 의하여 공유될수 없다. W2K의 신호기들은 전형적인 계수형신호기이다.

기다림시계는 Windows NT 4.0에서 제공하는 새로운 핵심부객체이다. 본질상 시계는 어떤 시간에 규칙적인 간격으로 신호를 낸다.

요약, 기본용어 및 복습문제

교착은 체계자원에 대하여 경쟁하거나 서로 통신하는 프로쎄스들의 모임이 페색된것이다. 조작체계가 한개 또는 그이상의 프로쎄스를 소멸하거나 한개 또는 그이상의 프로쎄스를 되돌이 추적하는것과 같은 그 어떤 례외적인 작업을 하지 않는이상 페색은 영구적이다. 페색은 재사용가능한 자원이나 소비되는 자원들을 포함한다. 소비되는 자원은 그것이 프로쎄스에 의해 차지될 때 파괴되는 자원이다. 실례로서 통보문과 입출력완충기의 정보를 들수 있다. 재사용가능한 자원은 입출력통로나 기억구역과 같이 사용해도 소모되거나 파괴되지 않는 자원이다.

교착을 취급하는 방법에는 일반적으로 예방, 검출, 피하기 등의 세가지가 있다. 교착예방은 교착의 필요조건들중의 하나가 성립하지 않도록 하는 방법으로 그것이 일어 나지 않도록 담보한다. 교착검출은 조작체계가 자발적으로 자원요청을 들어 주는 경우에 필요하다. 이때 조작체계는 주기적으로 교착을 검사하여야 하며 교착을 해제하기 위한 작업을 해야 한다. 교착회피는 매개 새로운 자원이 교착을 일으킬수 있는가를 확인하기 위하여 그것을 분석하며 교착이 불가능할 때에만 그것을 허가한다.

기본용어

은행업무알고리듬 순환기다림 소비되는 자원 교착	교착회피 교착검출 교착예방 유지 및 기다림	호상배제 선취권 재사용가능한 자원 고갈

복습문제

1. 재사용가능한 자원과 소비되는 자원의 실례를 드시오.
2. 교착을 가능하게 하는 세가지 조건은 무엇인가?
3. 교착을 발생시키는 네가지 조건은 무엇인가?
4. 유지 및 기다림조건은 어떻게 예방할수 있는가?
5. 비선취조건을 예방하는 두가지 방도에 대한 표를 작성하시오.
6. 순환기다림조건은 어떻게 예방할수 있는가?
7. 교착회피, 교착검출, 교착예방사이의 차이점은 무엇인가?

참 고 문 헌

교착과 관련한 고전적론문으로서 [COFF 71]과 [HOLT 72]가 있다. 좋은 개괄론문으로서 또한 [ISLO 80]이 있다. [CORB 96]에서는 교착예방에 대하여 충분히 취급하고 있다. [DIMI 98]은 교착에 대하여 친절하게 개괄해 주고 있다.

UNIX SVR 4, Solaris 2.x 그리고 Windows NT 4.0의 병행기구에 대해서는 각각 [GRAY 97], [GRAH 95] 그리고 [RICH 97]에서 취급하고 있다.

COFF71 Coffman, E.; Elphick, M.; and Shoshani, A. "System Deadlocks." *Computing Surveys,* June 1971.

CORB96 Corbett, J. "Evaluating Deadlock Detection Methods for Concurrent Soft-ware." *IEEE Transactions on Software Engineering,* March 1996.

DIMI98 Dimitoglou, G. "Deadlocks and Methods for Their Detection, Prevention, and Recovery in Modern Operating Systems." *Operating Systems Review,* July 1998.

GRAH95 Graham, J. *Solaris 2.x: Internals and Architecture.* New York: McGraw- Hill, 1995.

GRAY97 Gray, J. *Interprocess Communications in UNIX: The Nooks and Crannies.* Upper Saddle River, NJ: Prentice Hall, 1997.

HOLT72 Holt, R. "Some Deadlock Properties of Computer Systems." *Computing Surveys,* September 1972.

ISL080 Isloor, S., and Marsland, T. "The Deadlock Problem: An Overview." *Computer,* September 1980.

RICH97 Richter, J. *Advanced Windows.* Redmond, WA: Microsoft Press, 1997.

련 습 문 제

1. 제1절의 그림 6-2에 제시한 경로에 대한 서술과 류사하게 그림 6-3에 제시한 6개의 경로에 대하여 말로 설명하시오.

2. 그림 6-3에 반영한 정황에서는 교착이 발생될수 없다고 지적하였다. 이 지적의 정당성을 증명하시오.

3. 체계의 다음과 같은 순서상을 보자. 현재 결정되지 않은채로 대기렬을 지은 불충분한 요청들은 없다.

사용가능자원벡토르

r1	r2	r3	r4
2	1	0	0

프로쎄스	현재의 할당				최대요구				추가요구			
	r1	r2	r3	r4	r1	r2	r3	r4	r1	r2	r3	r4
P1	0	0	1	2	0	0	1	2				
p2	2	0	0	0	2	7	5	0				
p3	0	0	3	4	6	6	5	6				
p4	2	3	5	4	4	3	5	6				
p5	0	3	3	2	0	6	5	2				

ㄱ) 매개 프로쎄스가 얼마만큼 더 요청할수 있는가를 계산하시오. 그리고 추가요구라고 표시한 렬들에 적어 넣으시오.
ㄴ) 이 체계가 안정한 상태인가 아니면 불안정한 상태인가? 왜 그런가?
ㄷ) 이 체계가 현재 교착되여 있는가? 교착되여 있으면 왜 그렇고 교착되여 있지 않으면 왜 그런가를 설명하시오
ㄹ) 있다면 어떤 프로쎄스들이 교착되여 있는가? 아니면 교착이 될수 있는가?
ㅁ) (0, 1, 0, 0)일 때 P3으로부터 요청이 도착했다고 하면 그 요청이 즉시 안전하게 허가될수 있는가? 어떤 상태(교착된, 안정한, 불안정한)에서 모든 요청이 즉시에 허가된다면 어떤 프로쎄스들이 교착되여 있는가?(만일 교착되여 있다면) 아니면 교착될수 있는가?

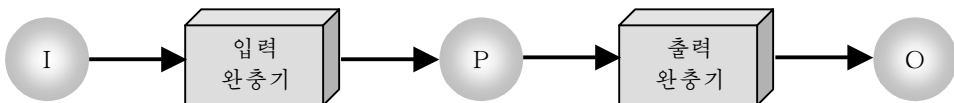



그림 6-14. 입출력완충체계

4. 다음의 자료에 대하여 제4절의 교착검출알고리듬을 적용하여 얻은 결과를 계산하시오.

사용가능 = (2 1 1 0)

$$요청 = \begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{pmatrix} \qquad 배경 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix}$$

5. 입출력완충체계(그림 6-14)는 두개의 완충기로 련결된 입력프로쎄스 I, 사용자프로쎄스 P 그리고 출력프로쎄스 O로 이루어 져 있다. 프로쎄스들은 자료를 동일한 크기의 블로크들로 교환한다. 이 블로크들은 프로쎄스들의 속도를 고려하여 입력 및 출력, 완충기들사이의 경계를 변경시키면서 디스크에 완충기억된다. 사용한 통신기본지령은 다음의 자원제약조건을 만족하도록 한다.

$$i + o \leq max$$

여기서 max = 디스크에 있는 블로크의 최대수
i = 디스크에 있는 입력블로크의 수
o = 디스크에 있는 출력블로크의 수

프로쎄스에 대하여 다음과 같은것을 알수 있다.

1) 환경이 자료를 공급하는이상 프로쎄스 I는 우발적으로 그것을 디스크에 입력한다(디스크공간을 사용할수 있는 조건에서).
2) 입력을 디스크에서 사용할수 있기만 하면 프로쎄스 P는 우발적으로 그것을 소비하며 매개 블로크입력에 대하여 디스크의 유한개의 자료를 출력한다(디스크공간을 사용할수 있는 조건에서).
3) 출력을 디스크에서 사용할수 있기만 하면 프로쎄스 O는 우발적으로 그것을 소멸한다.

이때 이 체계가 교착될수 있다는것을 증명하시오.

6. 문제 4에서 교착을 예방하면서도 프로쎄스들의 현재의 요구에 따라 입력 및 출력완충기들사이의 경계를 변동시키도록 하는 보충적인 자원제한조건을 제시하시오.

7. THE 다중프로그람작성체계[DIJK 68]에서 자기원통(디스크전에 나온 2차기억기)은 프로쎄스들의 속도에 의존하여 경계를 변경시키면서 입력완충기들, 처리구역들 그리고 출력완충기들로 나뉘여 진다. 자기원통의 현재 상태를 다음의 파라메터들에 의하여 특징 지을수 있다. 즉

max = 자기원통의 최대페지수
i = 자기원통의 입력페지수
p = 자기원통의 처리페지수
o = 자기원통의 출력페지수
reso = 출력용으로 예약된 최소페지수
resp = 처리용으로 예약된 최소페지수

자기원통의 용량을 초과하지 않으며 출력과 처리용으로 최소페지수가 항상 예약되여 있다는것을 담보하는데 필요한 자원의 제약조건을 식으로 쓰시오.

8. THE다중프로그람작성체계에서 페지는 다음과 같은 상태이행을 할수 있다. 즉

1) 빔->입력완충기 (입력생성)
2) 입력완충기->처리령역 (입력소비)
3) 처리구역->출력완충기 (출력생성)
4) 출력완충기->빔 (출력소비)
5) 빔->처리령역 (수속호출)
6) 처리령역->빔 (수속복귀)

ㄱ) 량 i, o 및 p에 의하여 이러한 변환의 효과를 정의하시오.

ㄴ) 입력프로쎄스, 사용자프로쎄스 및 출력프로쎄스에 대하여 문제 5에서 한 가정을 그대로 두면 그것들중의 하나가 교착에로 이끌어 갈수 있는가?

9. 아래에서 보여 주는것과 같이 세개의 프로쎄스에 할당된 총 150개의 단위의 기억기단위를 가진 체계를 고찰하자.

프로쎄스	최대	유지
1	70	45
2	60	40
3	60	15

다음의 매개 요청을 들어 주는것이 안전한가를 확인하기 위하여 은행업무알고리듬을 적용하시오. 만일 《예》이면 가능하다고 보증할수 있는 완료들의 순서를 표시하시오. 만일 《아니》이면 결과적인 할당표의 축도를 제시하시오.

ㄱ) 최대기억기요구가 60단위이고 초기기억기요구가 25단위인 네번째 프로쎄스가 도착한다.

ㄴ) 최대기억기요구가 60단위이고 초기기억기요구가 35단위인 네번째 프로쎄스가 도착한다.

10. 실제 생활에서 도움이 되도록 은행업무알고리듬을 평가하시오.

11. 흐름선알고리듬은 프로쎄스 P_0에 의하여 생성된 T형태의 자료요소흐름렬이 프로쎄스들인 $P_1, P_2, \cdots, P_{n-1}$의 순서렬을 통과해 나가도록 실현되여 있다. 이때 프로쎄스들의 순서를 따라 가면서 자료요소들에 대한 연산을 진행한다.

ㄱ) 부분적으로 소비되는 자료요소모두를 포함하는 일반화된 통보문완충기를 정의하시오. 그리고 다음의 형태로 프로쎄스 $P_i(0 \le i \le n-1)$에 대한 알고리듬을 작성하시오.

```
repeat
   receive  from  predecessor;
   consume  element;
   send  to  successor:
forever
```

P_0은 P_{n-1}에 의해 보내진 빈 요소들을 받는다고 하자. 알고리듬은 프로쎄스들이 완충기에 보관된 통보문에 대하여 직접 연산할수 있게 하여 복사가 필요없도록 해야 한다.

ㄴ) 공통완충기에 대해서는 프로쎄스들이 교착될수 없다는것을 증명하시오.

12. ㄱ) 세개의 프로쎄스가 한번에 한개만을 예약하고 해방할수 있는 네개의 자원단위들을 공유한다. 매개 프로쎄스는 최대로 두개단위를 요구한다. 이때 교착이 발생할수 없다는것을 증명하시오.

ㄴ) N개의 프로쎄스가 한번에 한개만을 예약하고 해방할수 있는 M개의 자원단위를 공유한다. 매개 프로쎄스의 최대요구는 M을 초과하지 않으며 모든 최대요구들의 합은 $M + N$보다 작다. 이때 교착이 발생할수 없다는것을 증명하시오.

13. 교착을 처리하는 다음의 방도들을 고찰하자. (1)은 은행업무알고리듬, (2)교착을 검출하고 스레드를 소멸하여 모든 자원을 해방, (3) 미리 모든 자원을 예약, (4) 스레드가 기다리기를 요구하면 스레드의 재시동과 모든 자원의 해방, (5)자원의 순서짓기, (6)교착의 검출과 스레드작업의 재연산

ㄱ) 교착에 대한 각이한 방법을 평가하는데서 사용할수 있는 한가지 기준은 가장 큰 병행성을 허가하는것이다. 다른 말로 말하면 교착이 없을 때 제일 많은 스레드들이 기다림이 없이 진행할수 있게 하는것이다. 우에서 지적한 매개 교착처리방법에 대하여 1부터 6까지의 등급순서를 매기시오. 여기서 1은 병행도가 제일 큰것이다. 작성한 순서짓기에 대하여 설명하시오.

ㄴ) 다른 기준은 효률이다. 달리 말하면 제일 적은 처리기간접소비시간을 요구하는것이다. 교착이 매우 드문 사건이라는것을 가정하고 방법들의 등급순서를 1부터 6까지로 매기시오. 여기서 1은 효과가 제일 큰것이다. 작성한 순서짓기에 대하여 설명하시오. 교착이 자주 발생한다고 하면 작성한 순서짓기가 변화되겠는가?

```
#define N             5            /*철학자수 */
#define LEFT          (i-1)  %N    /*I의 왼쪽 동료의 번호*/
#define RIGHT         (i+1)  %N    /*I의 오른쪽 동료의 번호*/
#define THINKING      0            /*철학자가 사고하고 있다.*/
#define HUNGRY                     1/*철학자가 포크를 쥐려고 한다.*/
#define EATING                     2/*철학자가 식사하고 있다.*/
typedef int semaphore;             /*신호기는 특수한 옹근수형이다.*/
int state[N];                      /*매 사람의 상태의 흔적을 보존하기 위한 배렬*/
*/
semaphore mutex = 1;               /*림계구역에 대한 호상배제*/
semaphore s[N];                    /*철학자당 한개의 신호기*/

void philosopher (int i)           /*철학자수, 0에서 N-1까지*/
{
    while  (TRUE)  {               /*부단히 반복한다.*/
        think();                   /*철학자가 사고하고 있다*/
        take_forks();              /*두개의 포크를 쥐거나 폐색*/
        eat();                     /*스퍼케티를 먹다.*/
```

```
        put_forks(i);                          /*상우의 두개의 포크를 도로 집다.*/
    }
}

void take_forks(int i)                         /*철학자수, 0에서 N-1까지*/
{
    wait(mutex);                               /*림계구역에 들어 가다.*/
    state[i] = HUNGRY;                         /*철학자 i가 굶고 있다는 사실을 기록*/
*/
    test(i);                                   /*두개의 포크를 주려고 시도하다*/
    signal(mutex);                             /*림계구역에서 나가다.*/
    wait(s[i]);                                /*포크를 쥐지 못했다면 폐색*/
}

void  put_forks(int i)                         /*철학자수, 0에서 N-1까지*/
{
    wait(mutex);                               /*림계구역에 들어 가다*/
    state[i] = THINKING;                       /*철학자가 식사를 끌내다.*/
    test(LEFT);                                /*왼쪽 동료가 지금 먹을수 있는가를 보다.*/
    test(RIGHT);                               /*오른쪽 동료가 지금 먹을수 있는가를 보다.*/
    siganal(mutex);                            /*림계구역에서 나가다.*/
}

void test(int i)                               /*철학자수, 0에서 N-1까지*/
{
   if (state[i] ==HUNGRY && state[LEFFT] !=EATING && state[RIGHT] != EATING)
   {
     state[i] = EATING;
     signal(s[i]);
    }
}
```

그림 6-15. 철학자식사문제에 대한 해답

14. 철학자식사문제에 대한 다음의 해답에 대하여 설명하시오. 식사하려는 철학자는 먼저 왼쪽의 포크를 쥔다. 만일 오른쪽의 포크도 쓸수 있다고 하면 그는 오른쪽의 포크를 쥐고 먹기 시작한다. 그렇지 않으면 그는 왼쪽의 포크를 다시 내려놓고 주기를 반복한다.

15. 이 문제는 철학자식사문제의 교묘성과 신호기를 사용하는 프로그람을 정확히 작성하는데서 제기되는 곤난성을 보여 준다. 그림 6-15에서는 최근에 나온 OS책들인 [TANE 97]과 [TANE 92]에 제시한 철학자식사문제에 대한 해답을 보여 주고 있다.

ㄱ) 이 해답에서 해결한 방도를 말로 설명하시오.

ㄴ) 저자들은 말하고 있다. 즉 《해답은 … 정확하며 임의의 인원수의 철학자들에 대하여 최대의 병렬화를 보장한다.》 그러나 이 해답이 교착을 예방한다고 하지만 고갈이 가능하므로 정확하지는 못하다. 상반되는 실례에 의하여

이것을 증명하시오. 요령: 다섯명의 철학자가 있는 경우를 고찰하자. 그들은 단식한 철학자들이라고 하자. 즉 그들은 거의 사고함이 없이 시간을 보낸다. 어떤 철학자가 한판의 식사를 끝내자마자 그는 즉시에 배고파한다. 이때 두 사람은 현재 식사를 하고 있고 다른 세 사람은 봉쇄되여 배고파하는 구성을 고찰하시오.

16. 두 부류의 철학자들이 있다고 하자. 한 부류는 항상 먼저 왼쪽의 포크를 쥐며(왼손잡이) 다른 부류는 항상 먼저 오른쪽의 포크를 쥔다(오른손잡이). 왼손잡이의 동작을 그림 6-11에서 정의하였다. 오른손잡이의 동작은 다음과 같다.

```
begin
        repeat
        think;
        wait ( fork [ ( i+1) mod 5 ] );
        wait ( fork [ i ] );
        eat;
        signal ( fork [ i ] );
        signal ( fork [ ( i+1) mod 5 ] );
      forever
    end;
```

이때 다음의것을 증명하시오.

ㄱ) 적어도 매개 부류에서 한사람씩의 왼손잡이들과 오른손잡이들이 끼운 임의의 좌석배치를 하면 교착을 피할수 있다.

ㄴ) 적어도 매개 부류에서 한사람씩의 왼손잡이들과 오른손잡이들이 끼운 임의의 좌석배치를 하면 고갈을 예방한다.

제 3 편. 기억기

제 3 편의 중심

조작체계설계에서 가장 어려운 문제의 하나는 기억기관리문제이다. 기억기의 가격이 급격히 떨어 지고 결과 현대적인 콤퓨터들에서 주기억기의 크기가 증가하여 기가바이트 범위에 이르렀지만 아직도 능동프로쎄스나 조작체계가 요구하는 모든 프로그람이나 자료구조를 충당할만큼 충분하지 못하다. 따라서 조작체계의 중심적인 과제의 하나는 기억기관리인바 여기에는 2차기억기로부터의 자료블로크의 들여오기와 교체내기가 포함된다. 한편 기억기입출력은 저속동작으로서 그 속도는 처리기의 명령주기시간에 비하여 해마다 더욱더 떨어 지고 있다. 처리기 또는 처리기들의 차지상태를 유지하고 따라서 효률을 높이기 위해서는 조작체계가 기억기입출력의 성능에 미치는 영향이 최소화되도록 교체넣기와 교체내기에 드는 시간을 세밀하게 조절하여야 한다.

제 3 편의 안내

제 7 장. 기억기관리

제7장에서는 기억기관리에서 사용하는 기본 기구를 개괄한다. 우선 기억기관리방안들에서 제기되는 기초적인 요구를 요약한다. 다음으로 기억기분할의 사용수법을 소개한다. 이 수법은 핵심부부분 기억기관리와 같은 특별한 경우를 제외하고는 그리 많이 쓰지 않는다. 하지만 기억기분할을 개괄함으로써 기억기관리에 포함되는 설계상의 여러 문제점들을 해명한다. 이 장의 나머지 부분에서는 실제상 모든 기억기관리체계들의 기본적인 구성방법인 두가지 수법 즉 페지화와 토막화에 대하여 서술한다.

제 8 장. 가상기억기

가상기억기는 페지화 또는 페지화와 토막화를 결합하여 사용하는것을 기초로 한것으로서 현대적인 콤퓨터들에서 거의나 보편적인 기억기관리방법이다. 가상기억기는 응용프로그람의 프로쎄스들에 대하여 투명한 방안이며 매개 프로쎄스가 그의 배치령역에서 제한 없는 주기억을 가진것처럼 동작하는 방안이다. 이것을 실현하기 위하여 조작체계는 디스크상에 매개 프로쎄스에 대한 가상주소공간을 만든다. 가상기억기의 한 부분은 필요할 때 실제의 주기억기에로 들어 온다. 이러한 방법으로 많은 프로쎄스들이 상대적으로 작은 주기억용량을 공유할수 있다. 가상기억기를 효과적으로 동작시키기 위하여 하드웨어기구는 가상주소와 실제주소사이의 주소변환과 같은 기본적인 페지화와 토막화기능을 수행하여야 한다. 제 8장은 이러한 하드웨어기구들을 개괄하는것으로부터 시작한다. 이 장의 마지막부분에서는 가상기억기와 관련된 조작체계설계의 문제점들을 고찰한다.

제 7 장. 기억기관리

단일프로그람처리체계에서 주기억기는 두 부분으로 나누어 진다. 한 부분은 조작체계(상주된 감시기와 핵심부)가 차지하는것이고 한 부분은 현재 실행되는 프로그람이 차지하는것이다. 다중프로그람처리체계에서는 기억기의 《사용자》부분이 여러개의 프로쎄스를 실행하도록 보다 더 세분화되여야 한다. 이 세분화과제를 조작체계가 동적으로 수행하는데 이것을 **기억기관리**라고 한다.

효과적인 기억기관리는 다중프로그람처리에서 보다 사활적이다. 기억기에 몇개의 프로쎄스밖에 없다면 많은 시간동안 모든 프로쎄스들이 입출력을 위해 기다리게 되며 처리기는 놀고 있는 상태에 있게 된다. 그러므로 기억기는 가능한껏 더 많은 프로쎄스들이 기억기에 들어 가도록 효률적으로 배정할것을 요구한다.

이 장에서는 먼저 기억기를 충분히 관리하는데서 제기되는 요구사항들을 설명한다. 다음에 현재 사용되는 여러가지 단순한 기억기관리방안들을 고찰함으로써 기억기관리기술에 접근해 간다. 여기서 주목하는것은 프로그람이 실행되자면 그것이 주기억기에 적재되여야 한다는 요구사항이다. 이것에 대한 해설은 기억기관리의 기본적인 몇가지 원리를 소개하는 방법으로 진행한다.

제 1 절. 기억기관리의 요구

기억기관리와 관련한 여러가지 기구들과 방책들을 조사해 보면 기억기관리가 만족해야 할 요구사항들을 [LIST93]에서는 다음의 5가지로 제기하고 있다. 즉

- 재배정
- 보호
- 공유
- 론리조직
- 물리조직

재배정

다중프로그람처리에서 사용가능한 주기억기는 일반적으로 여러개의 프로쎄스들사이에서 공유된다. 대표적으로 프로그람작성자는 프로그람이 실행중에 있을 때 다른 프로그람이 주기억기에 상주하겠는지를 미리 알수 없다. 더우기 실행할 준비프로쎄스들의 큰 집결소를 줌으로써 처리기의 사용률을 최대로 높이기 위하여 능동프로쎄스들을 주기억기에 교체넣기하거나 주기억기에서 교체내기하여야 한다. 일단 프로그람이 디스크에로 교체내기되였으면 다음에 그것이 거꾸로 교체넣기할 때 이전과 같은 주기억구역에 배치되여야 한다고 결론하기는 대단히 어렵다. 대신에 프로쎄스를 기억기의 다른 구역에 **재배정**할 필요가 있다.

이때 프로그람이 어디에 위치하는지 앞질러 알수 없으며 교체로 인하여 프로그람이 주기억기안에서 사방으로 이동할수 있다. 이와 같은 사정은 그림 7-1에 보여 준것과 같은 주소지정과 관련된 몇가지 기술적문제를 발생시킨다. 이 그림은 프로쎄스형태를 보여주고 있다. 간단히 고찰하기 위하여 프로쎄스형태가 주기억기에서 련속구역을 차지한다고 가정한다. 명백하게 조작체계는 프로쎄스조종정보와 실행탄창의 위치, 프로쎄스용프로

그람의 실행을 시작하는 입구점의 위치를 알아야 할것이다. 조작체계가 기억기를 관리하고 프로쎄스를 주기억기에로 끌어 들여야 하므로 주소들을 쉽게 얻어 낼수 있다. 더우기 처리기는 프로그람안에서 기억기의 참조문제를 취급하여야 한다. 갈래명령안에는 다음번에 실행될 명령을 참조하기 위한 주소가 들어 있다. 자료참조명령에는 참조되는 자료에 대한 바이트 혹은 단어길이주소가 포함되여 있다. 한편 처리기하드웨어와 조작체계쏘프트웨어인 프로그람코드안에 있는 기억기참조값들을 주기억기안에서 현재 프로그람의 위치를 반영하는 실제적인 기억기물리주소값들로 변환할수 있어야 한다.

보호

매개 프로쎄스들은 우연적이든 계획적이든 다른 프로쎄스에 의한 간섭으로부터 보호되여야 한다. 그러면 다른 프로쎄스의 프로그람들은 허가 없이 읽거나 쓰기의 목적으로 프로쎄스안의 기억기위치를 참조할수 없다. 한마디로 재배정요구들을 만족시키면 그에 따라 보호의 요구를 만족시키기가 힘들어 진다. 주기억기안에서 프로그람의 위치는 예견할수 없으므로 보호를 확증하기 위해 콤파일시에 절대주소를 조사한다는것은 불가능하다. 더우기 대다수 프로그람작성언어들은 실행할 때 동적으로(실례로 배렬첨수지정자나 자료구조에 대한 지시자를 계산하는것으로) 주소계산을 진행한다. 프로쎄스에 의하여 발생되는 모든 기억기참조들은 프로쎄스에 배정된 기억공간만 확실히 참조하기 위하여 실행할 때 검사하여야 한다. 그리하여 여기서는 재배정을 지원하고 보호의 요구를 지원하는 기구를 고찰한다.

보통 사용자프로쎄스는 조작체계의 부분이나 프로그람 또는 자료에 접근할수 없다. 또한 한 프로쎄스안의 프로그람을 보통 다른 프로쎄스안의 명령에로 갈래를 일으킬수 없다. 한 프로쎄스안의 배치를 특별히 하지 않는한 프로그람은 다른 프로쎄스의 자료에 접근할수 없다. 처리기는 실행점에서 명령을 취소할수 있어야 한다.

기억기보호요구는 조작체계(쏘프트웨어)보다 처리기(하드웨어)에 의하여 만족되여야 한다. 이것은 프로그람에서 진행되는 모든 기억기참조들을 조작체계가 미리 예상할수 없기때문이다. 비록 그러한 예상을 할수 있다고 해도 매개 프로그람에 대하여 미리 있을수 있는 기억기참조위반을 막는다는것은 대단한 시간랑비일것이다. 참조하는 명령을 실행할 때 기억기참조(자료접근이나 갈래)를 허용할수 있다. 이것을 실현하기 위해서 처리기하드웨어는 그런 능력을 가지고 있어야 한다.

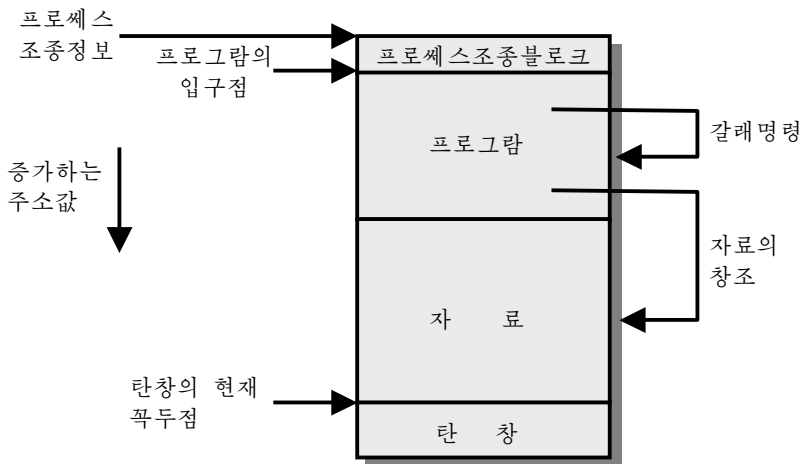



그림 7-1. 프로쎄스에 대한 주소지정의 요구

공유

어떤 보호기구든지 여러개의 프로쎄스들이 주기억기의 동일한 부분을 호출할수 있게 해야 한다. 실례로 여러개의 프로쎄스들이 동일한 프로그람을 실행하고 있다면 매개 프로쎄스들이 프로그람의 한개 사본에 접근하는것이 자기의 사본에 각각 접근하는것보다 편리하다. 어떤 과제상에서 서로 협동하는 프로쎄스들은 동일한 프로쎄스에 대한 접근을 공유할 필요가 있을수 있다. 그러므로 기억기관리체계는 필수적인 보호를 보장하면서도 공유하는 기억구역에 대한 제한된 접근을 허용하여야 한다. 이로부터 재배정지원공유능력을 지원하는데 사용되는 기구를 보게 될것이다.

론리조직

콤퓨터체계에서 주기억기는 거의나 변함 없이 바이트 또는 단어순서들을 구성하는 선형적인 또는 1차원적인 주소공간으로 조직된다. 2차기억기도 물리적인 준위에서 거의 류사하게 조직된다. 이러한 조직방법은 실제적인 기계하드웨어에 밀접하게 반영되기는 하지만 프로그람을 구축하는 전형적인 방법에 대응하지는 못한다. 대부분의 프로그람들은 모듈들로 조직화하는데 그것들중의 일부는 변경할수 없는것이고 (읽기만 하거나 실행만 하는것) 또다른 일부는 변경할수 있는 자료를 포함하고 있다. 만일 조작체계와 콤퓨터하드웨어가 사용자프로그람과 자료를 몇가지 종류의 모듈형태로 효과적으로 취급할수 있다면 여러가지 우점을 발휘할수 있다. 즉

1. 모듈들은 실행할 때 한 모듈로부터 체계에 의해 파생되는 다른 모듈에 대한 모든 것을 참조하면서 독립적으로 작성되고 콤파일될수 있다.
2. 적당한 간접소비시간을 소비하면서 각이한 모듈들에 각이한 등급의 보호(읽기만 하거나 실행만 하는것)를 걸수 있다.
3. 모듈들이 프로쎄스들사이에서 공유할수 있는 기구들을 적용할수 있다. 모듈준위에서 공유를 실현하는것이 가지는 우점은 이것이 사용자의 문제고찰방법에 대응하고 있고 이로부터 사용자가 희망하는 공유를 쉽게 명시할수 있는것이다.

이 요구사항들을 가장 쉽게 만족시키는 도구가 바로 토막화인데 이것은 이 장에서 취급하는 기억기관리수법의 하나이다.

물리조직

제1장 제5절에서 서술한바와 같이 콤퓨터기억기는 최소한 주기억기와 2차기억기라고 부르는 두개의 준위로 구성된다. 주기억기는 상대적으로 원가가 높고 호출속도가 빠르다. 주기억기는 또한 휘발성이다. 즉 영구기억기가 아니다. 2차기억기는 주기억기에 비하여 속도가 뜨고 값이 눅으며 보통은 비휘발성이다. 이렇게 대용량 2차기억기가 프로그람이나 자료를 오랜 기간 기억하는 반면에 보다 작은 용량의 주기억기는 현재 사용하고 있는 동안만 프로그람과 자료를 유지한다.

2준위방안에서는 주기억기와 2차기억기사이의 정보흐름을 조작하는데 체계가 중요하게 관여한다. 이것을 해결할것을 개별적인 프로그람작성자에게 부과시킬수도 있으나 이것은 다음의 두가지 리유로 하여 실현할수도 없고 희망할수도 없다. 즉

1. 프로그람과 그것의 자료용으로 사용가능한 주기억기가 충분하지 못할수 있다. 그러한 경우에 프로그람작성자는 실천에서 겹쳐놓기라고 알려 진 수법을 사용하는데 겹쳐놓기에서는 필요할 때 모듈들을 안팎으로 절환하는 주프로그람과 자료는 각이한 모듈들이 동일한 기억구역을 할당받을수 있게 조직된다. 콤파일러도구의

방조가 있다해도 겹쳐놓기프로그람작성은 프로그람작성자의 시간을 랑비한다.

2. 다중프로그람처리환경에서 프로그람작성자는 프로그람을 작성할 때 얼마나 큰 공간을 사용할수 있고 또 그 공간이 어디에 있는지 알수 없다.

이로부터 기억기의 2준위사이에서 체계가 움직이도록 하는것은 명백하다. 이 과제가 바로 기억기관리의 본질이다.

제 2 절. 기억기의 분할

기억기관리의 원리적조작은 프로그람을 처리소자가 실행하도록 그것을 주기억기에 끌어 들이는것이다. 거의 모든 현대적인 다중프로그람처리체계에서 이것은 가상기억기라고 하는 정교한 방안을 포함한다. 가상기억기는 바꾸어 말하면 토막화와 페지화라는 두가지 기본 수법중의 한가지 또는 두가지의 사용에 기초한것이다. 가상기억기의 수법을 고찰하기전에 가상기억기를 포함하지 않는 보다 간단한 수법을 먼저 고찰하여 기초를 준비하도록 한다(표7-1). 그러한 수법들중의 하나가 바로 분할법인데 이것은 여러가지 종류의 시대에 뒤떨어 지지 않는 조작체계에서 사용되고 있다. 다른 두가지 수법은 단순페지화와 단순토막화인데 그자체로서는 쓰지 않는다. 어쨌든 가상기억기를 고찰하면 가상기억기에 대한 표상이 명백해 질것이다.

고정분할법

대부분의 기억기관리방안들에서는 조작체계가 주기억기의 고정된 부분을 차지하고 나머지 부분을 여러개의 프로쎄스들이 사용한다고 가정할수 있다. 이러한 사용가능한 기억기를 관리하는 가장 간단한 방안이 그것을 고정경계를 가진 령역으로 분할하는것이다.

표 7-1. 기억기관리수법

수법	설명	우점	결함
고정분할법	체계가 발생할 때 주기억기를 여러개의 정적분할구역으로 나눈다. 프로쎄스는 크기가 같거나 더 큰 분구에 적재될수 있다.	실현이 간단하고 조작체계의 간접소비시간이 작다.	내부쪼각들때문에 기억기사용이 비효률적이고 능동프로쎄스의 최대수가 고정된다.
동적분할법	분할구역들이 동적으로 만들어 지므로 매개 프로쎄스는 그 프로쎄스와 꼭 같은 크기의 분할구역에 적재된다.	내부쪼각이 없고 주기억기의 사용이 보다 효률적이다.	외부쪼각을 계수해야 하므로 처리기의 사용이 비효률적이다.
단순페지화법	주기억기를 크기가 동일한 많은 프레임으로 나눈다. 매개 프로쎄스는 프레임과 같은 길이를 가지는 동일한 크기의 많은 페지들로 나눈다. 프로쎄스는 가능한껏 린접하지만 필수적으로 린접하지는 않는 프레임들에 모든 페지들을 적재하는식으로 적재된다.	외부쪼각이 없다.	적은 량의 내부쪼각이 있다.
단순토막화법	매개 프로쎄스는 많은 토막들로 나눈다. 프로쎄스는 린접해야 할 필요가 없는 동적인 분할구역들에 모든 토막들을 적재하는 식으로 적재된다.	내부쪼각이 없다.	기억기사용률을 개선하고 동적분할법에 비하여 간접소비시간이 줄어 든다.

표계속

가상기억기 페지화법	프로쎄스의 모든 페지들을 적재할 필요가 없는것을 제외하고는 단순페지화법과 같다. 요구되는 비상주페지들은 후에 자동적으로 끌어 들인다.	외부쪼각이 없다. 다중프로그람처리의 등급이 높고 가상주소공간이 크다.	복잡한 기억기관리로 인한 간접소비시간이 있다.
가상기억기 토막화법	프로쎄스의 모든 토막들을 적재할 필요가 없는것을 제외하고는 단순토막화법과 같다. 요구되는 비상주토막들은 후에 자동적으로 끌어 들인다.	내부쪼각이 없고 다중프로그람처리의 등급이 높다. 가상주소공간이 크다. 보호와 공유를 지원한다.	복잡한 기억기관리로 인한 간접소비시간이 있다.

분할구역크기

그림 7-2에서는 두가지 서로다른 고정분할법의 실례를 보여 주었다. 한가지는 같은 크기의 분할구역을 사용하는것이다. 이러한 경우에 분할구역의 크기보다 작거나 그것과 크기가 같은 프로쎄스는 임의의 사용가능한 분할구역에 적재될수 있다. 만일 모든 분할구역들이 다 찼거나 프로쎄스가 준비 또는 실행상태에 있지 않으면 조작체계는 임의의 분할구역의 프로쎄스를 교체하여 내 보내고 다른 프로쎄스를 적재하는데 그러한 작업은 처리기가 수행한다.

ㄱ	ㄴ
조작체계 8M	조작체계 8M
8M	2M
8M	4M
8M	6M
8M	8M
8M	8M
8M	12M
8M	16M

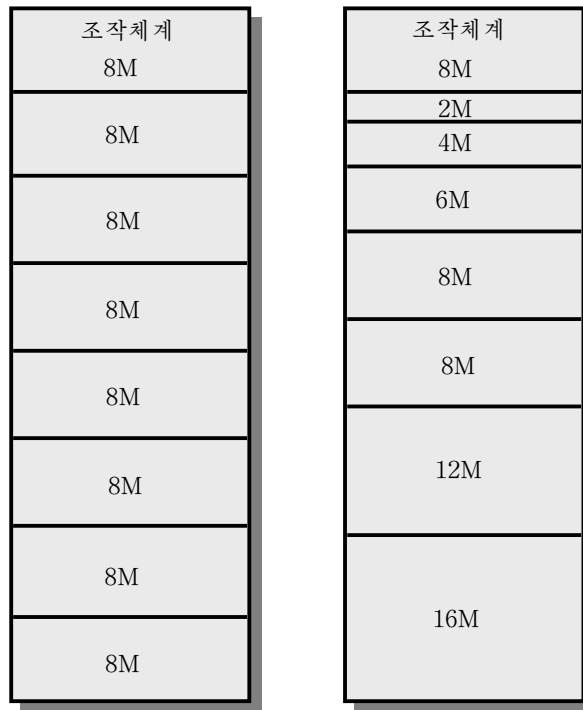



그림 7-2. 64Mbyte 기억기의 고정분할실례
ㄱ-동일크기분할, ㄴ-비동일크기분할

동일한 크기의 고정분할구역을 사용하는데는 두가지 난점이 있다. 즉

- 동일한 크기의 분할구역보다 지내 클수 있다. 이때 프로그람작성자는 겹쳐놓기법을 사용하여 프로그람을 설계해야 하는데 이렇게 되면 어떤 한 순간에 주기억기에는 프로그람의 한 부분만이 있어도 된다. 현재 없는 모듈이 필요할 때 사용자 프로그람은 거기에 프로그람이나 또는 자료가 있던지간에 겹쳐놓기를 하면서 프로그람의 분할구역에 그 모듈을 적재하여야 한다.
- 주기억기관리가 대단히 비효률적이다. 어떤 프로그람이 얼마나 작은지에는 관계없이 옹근 분할구역을 차지한다. 실례에서 보면 프로그람의 길이가 2MB보다 작을수 있지만 그것이 교체되여 들어올 때는 역시 8MB의 분할구역을 차지한다. 적재된 자료블로크가 분할구역보다 더 작아서 분할구역에 랑비되는 기억공간이 있게 되는 현상을 **내부쪼각화**라고 한다.

이러한 두가지 문제는 비동일크기분할법(그림 7-2ㄴ)을 써서 줄일수 있지만 완전히 해결할수는 없다. 이 실례에서 16MB만큼 큰 프로그람들은 겹쳐놓기가 없이 조정할수 있다. 크기가 8MB보다 작은 분할구역들은 크기가 보다 작은 프로그람들로 하여금 내부쪼각이 적게 되도록 조정할수 있다.

배치알고리듬

동일크기분할에서는 기억기상에서 프로쎄스의 배치가 단순하다. 임의의 사용가능한 분할구역이 있으면 프로쎄스는 그것에 적재될수 있다. 모든 분할구역의 크기가 같기때문에 어느 분할구역이 사용되든 상관이 없다. 만일 모든 분할구역들이 실행준비상태가 아닌 프로쎄스들로 꽉 찼다면 이 프로쎄스들중의 하나가 새로운 프로쎄스를 위한 마당을 마련하기 위해 교체되여 나가야 한다. 어느것을 교체해 내 보내는가가 바로 일정작성결정인데 이것에 대하여서는 제4편에서 보게 된다.

비동일크기분할에서는 프로쎄스를 분할구역에 할당하는데 두가지 방법이 있다. 매개 프로쎄스를 할당하는 가장 단순한 방법은 그것이 들어 맞는 분할구역들가운데서 가장 작은것에 할당하는것이다.[1] 그러한 경우에 그 분할구역을 쓰기로 예정되여 있는 교체되여 나간 프로쎄스들을 유지하기 위하여 매개 분할구역에 대한 일정작성대기렬이 요구된다 그림 (7-3ㄱ). 이 방법의 우점은 프로쎄스들이 항상 분할구역에서 랑비되는 기억공간(내부쪼각)을 최소화하는 방향에서 할당된다는것이다.

이 방법이 개별적인 분할구역의 시점에서는 최적인것처럼 보이지만 체계전체의 시점에서 보면 최적이 못된다. 실례를 들어 그림 7-2ㄴ에서 보면 어떤 시점에서 크기가 12M~16M사이에 프로쎄스가 없는 경우를 생각할수 있다. 그러한 경우 16M의 분할구역은 그보다 작은 프로쎄스가 그것에 할당할수 있었다해도 사용하지 않은채로 남아 있다. 이때 더 나은 방법은 모든 프로쎄스들에서 하나의 대기렬을 사용하는것이다(그림 7-3ㄴ).
프로쎄스를 기억기에 적재할 때 프로쎄스를 담게 될 사용가능한 분할구역이 선택된다. 분할구역들이 모두 차지된다면 교체결정을 하여야 한다. 좋기는 들어 오는 프로쎄스를 담게 될 가장 작은 분할구역을 교체하여 내 보내는것이다. 또한 우선권과 같은 다른 요인들과 페색된 프로쎄스들과 준비된 프로쎄스 등을 교체해 내기 위한 좋은 방법도 생각해 볼수 있다.

---

[1] . 이것은 프로쎄스가 요구하는 기억기의 최대량을 안다고 가정한다. 그런데 이것은 항상 있는 경우가 아니다. 프로쎄스가 얼마나 큰것인지 알수 없다면 겹쳐놓기방안이나 가상기억기를 사용하는수밖에 다른 방법이 없다.

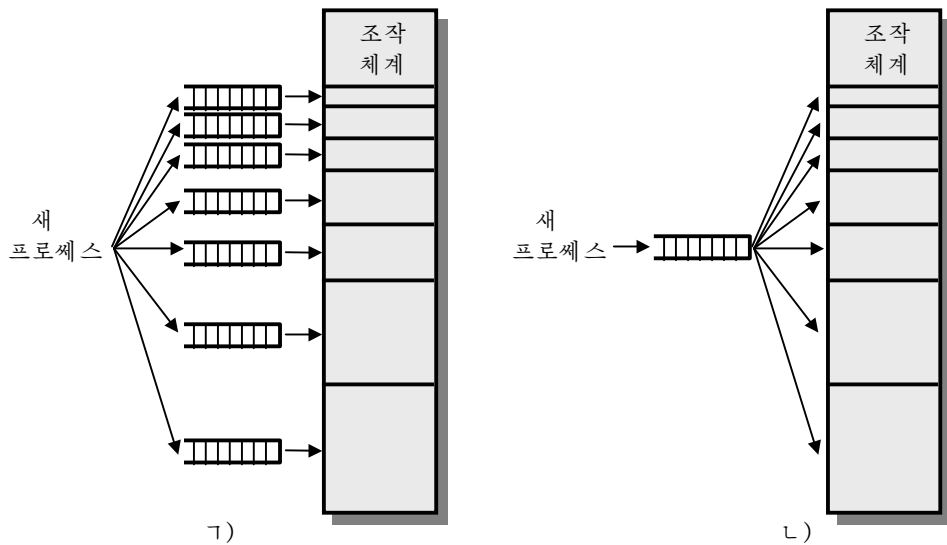



그림 7-3. 고정분할법에 의한 기억기할당
ㄱ-분할구역당 한개의 프로쎄스대기렬, ㄴ-단일대기렬

비동일크기분할법은 고정크기분할법에 비하여 유연성을 보장한다. 또한 고정분할방안들은 비교적 단순하고 최소한의 조작체계쏘프트웨어와 간접소비시간을 요구한다고 말할수 있다. 한편 다음과 같은 결함을 가지고 있다. 즉

- 체계발생시에 명시된 분할구역의 수는 체계안에 있는 응용프로쎄스(중단되지 않은)의 수를 제한한다.
- 체계발생시에 분할구역의 크기가 미리 정해지기때문에 작은 일감들은 분할구역공간을 효률적으로 사용할수 없다. 모든 일감들의 기본적인 기억기요구사항들이 미리 알려 진 환경에서는 이 수법이 적합할수 있으나 대부분의 경우에는 효률적이지 못하다.

고정분할법은 현재는 거의 사용하지 않는다. 사용한 성공적인 조작체계의 한가지 실례로서는 이전에 나온 IBM대형콤퓨터조작체계 OS/MFT(과제수가 고정된 다중프로그람처리체계)를 들수 있다.

동적분할법

고정분할법의 몇가지 난점들을 극복하기 위하여 동적분할법이 개발되였다. 이 방법은 보다 정교한 기억기관리수법들에 의하여 다시 크게 교체되였다. 이 수법을 사용한 중요한 조작체계는 IBM의 대형콤퓨터조작체계OS/MVT(과제수가 변하는 다중프로그람처리체계)이다.

동적분할에서는 할당구역의 길이와 개수가 변한다. 프로쎄스가 주기억기에 들어 올 때 분할구역은 정확히 프로쎄스가 요구하는 크기만큼 배정되고 더 되지 않는다. 그림 7-4에서는 64Mbyte의 주기억기를 사용하는 실례를 보여 주고 있다. 초기에 주기억기는 조작체계를 제외하고 모두 비여있다(ㄱ). 처음 3개의 프로쎄스가 적재되는데 조작체계가 끝나는곳에서 시작하여 매개 프로쎄스에 충분한 공간을 차지하고 있다(ㄴ, ㄷ, ㄹ). 이것은 기억기의 마지막 부분에 4번째 프로쎄스를 넣기는 너무 작은《구멍》을 남겨 놓고 있

다. 어떤 점에서 기억기안에 있는 프로쩨스들은 준비상태에 있지 않다. 조작체계는 프로쩨스 2를 교체해 내 보내며(ㅁ) 새로운 프로쩨스 4를 넣는데 충분한 공간을 마련한다. 프로쩨스4가 프로쩨스 2보다 작기때문에 또다른 작은 구멍이 생겨 난다. 후에 주기억기안에 있는 프로쩨스들이 준비되지 않은 상태에 도달하지만 준비–중단상태에 있는 프로쩨스 2는 사용가능하다. 프로쩨스 2에서는 기억기의 공간이 충분하다. 조작체계는 프로쩨스1을 교체해 내보내고(ㅅ) 프로쩨스 2를 도로 교체해 들여 온다(ㅇ).

이 실례에서는 이 방법이 시작은 좋게 뗐지만 결과적으로는 기억기에 수많은 작은 구멍들이 있는 상태에 이르게 된다는것을 보여 준다. 시간이 지날수록 기억기는 더욱더 쪼각나고 기억기관리는 종말에 이르게 된다. 이러한 현상을 **외부쪼각화**라고 하는데 이것은 모든 분할구역에 대하여 바깥쪽의 기억기가 점차적으로 쪼각나는데 기인된다. 이것은 앞에서 고찰한 내부쪼각화와 대조되는것이다.

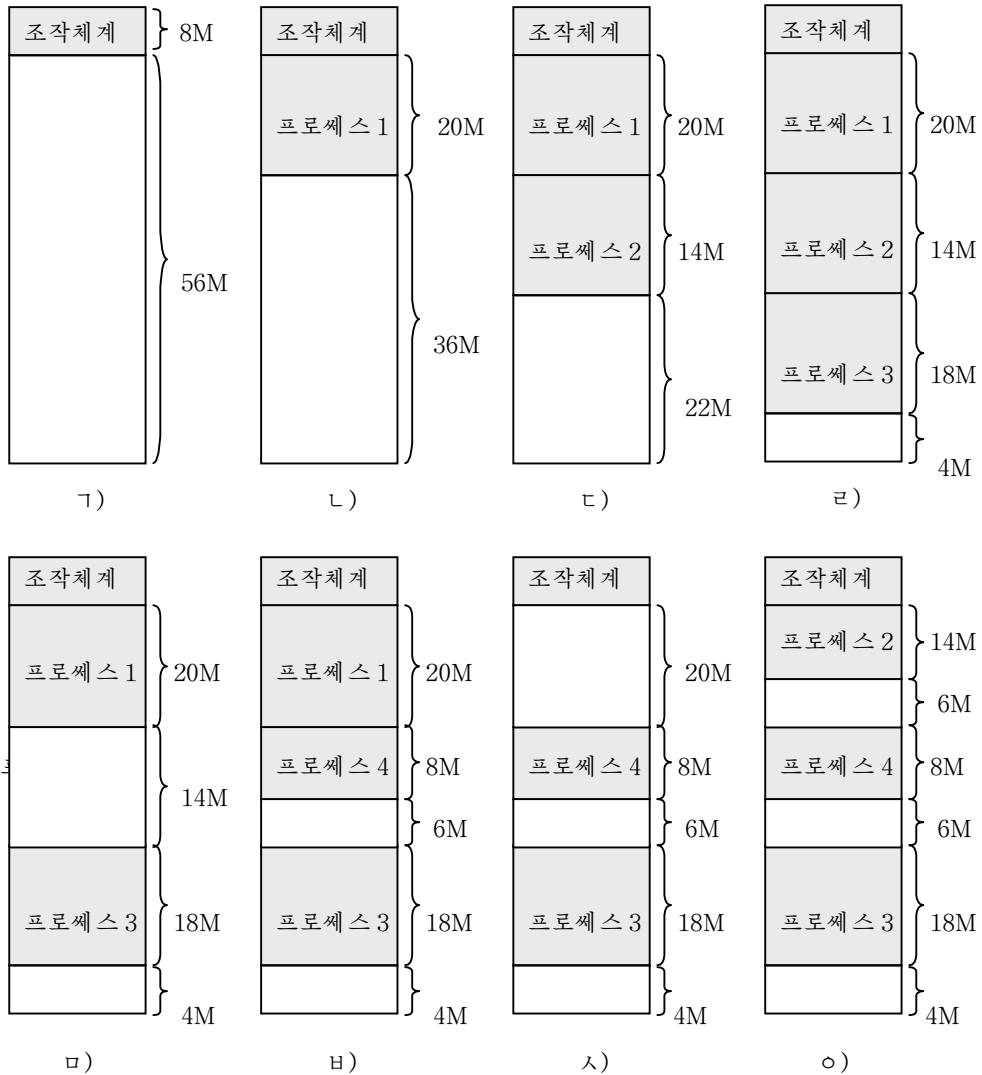



그림 7-4. 동적분할효과

외부쪼각화를 극복하기 위한 한가지 수법이 바로 **조이기**이다. 즉 시간이 경과함에 따라 조작체계는 프로쎄스들을 그것이 린접되여 있도록 이동하며 그것으로 하여 모든 자유기억구역들이 한개의 블로크에 모이게 된다. 실례를 들어 그림 7-4 ㅇ에서 조이기는 한개의 블로크에 16MB길이의 빈 기억기가 생기는 결과를 초래한다. 이것은 프로쎄스를 추가하여 적재하는데 충분할수 있다. 압축의 난점은 그것이 시간을 소비하는 수속이라는것과 이로부터 처리기의 시간이 랑비되는것이다. 조이기는 동적인 재배정능력을 필요로 한다. 즉 프로그람에서 기억기참조들을 무효화함이 없이 주기억기의 한 구역에서 다른 구역에로 프로그람을 옮길수 있어야 한다(부록 7-ㄱ를 보라).

배치알고리듬

기억기조이기에 시간이 들므로 조작체계설계자는 프로쎄스를 기억기에 어떻게 할당하겠는가(구멍을 어떻게 메우겠는가)를 판단하는데 재치가 있어야 한다. 프로쎄스를 주기억기에 적재하거나 교체할 때 충분한 크기를 가진 자유기억기블로크가 한개이상 있다면 조작체계는 어느 자유블로크를 배치하겠는가를 결정해야 한다.

최적적합, 앞방향적합, 뒤방향적합 등 세가지 배치알고리듬을 사용할수 있다. 이것들은 물론 끌어 들이는 프로쎄스와 크기가 같거나 보다 큰 주기억기안의 자유블로크들가운데서 하나를 선택하는것으로서 제한된다. 최적적합형은 요구되는것과 크기상 가장 가까운 블로크를 선택하는것이다. 앞방향적합형은 기억기의 첫 머리에서부터 조사하여 충분히 큰 첫 사용가능한 블로크를 선택하는것이다. 뒤방향적합형은 마지막으로 배치된 위치에서부터 기억기를 조사하여 충분히 큰 다음의 사용가능한 블로크를 선택하는것이다.

그림 7-5 ㄱ에서는 여러번의 배치와 교체내기조작을 하고난 후의 기억기구성상태를 실례로 보여 주고 있다. 사용된 마지막 블로크가 22MB였는데 그것으로부터 14MB의 분할구역이 창조되였다. 그림 7-5에서는 16MB의 배정요청을 만족시키는데서 세가지 배치알고리듬사이의 차이점을 보여 주고 있다. 최적적합알고리듬은 사용가능한 블로크들의 전체 목록을 탐색하여 2MB의 쪼각을 남기는 18MB 블로크를 사용한다. 앞방향적합알고리듬은 6MB의 쪼각을 만들고 뒤방향적합알고리듬은 20MB의 쪼각을 남긴다.

이 방법들중의 어느것이 제일 좋은가는 집행되는 프로쎄스교체의 정확한 순서와 이 프로쎄스들의 크기에 관계된다. 한편 몇가지 일반적인 주해를 붙일수 있다([BREN89], [SHOR75], [BAYS77]을 보라.). 앞방향적합알고리듬은 가장 단순할뿐아니라 일반적으로 가장 좋고 속도도 제일 높다. 뒤방향적합알고리듬은 앞방향적합알고리듬에 비하여 조금 나쁜 결과를 내는 경향이 있다. 뒤방향적합알고리듬은 더 자주 기억기의 끝부분에 있는 자유블로크로부터 배정을 해 나간다. 결과 흔히 기억기의 끝부분에 나타나는 자유기억기의 가장 큰 블로크가 작은 쪼각들로 인차 깨여 진다. 그리하여 뒤방향적합알고리듬에서는 조이기가 더 자주 요구될수 있다. 한편 앞방향적합알고리듬은 매개 순차적인 앞방향적합경로에서 찾을 필요가 제기되는 작은 자유할당구역들로 전방끝부분을 분산시킬수 있다. 최적적합알고리듬은 그 이름에도 불구하고 보통 결과가 제일 나쁜것으로 된다. 이 알고리듬은 요구를 만족시키는 가장 작은 블로크를 탐색하기때문에 그것은 가능한 제일 작은 쪼각이 뒤에 남아 있다는것을 담보할수 있다. 매 기억기요구가 항상 가장 작은 량의 기억기를 소비할것을 요구하지만 결과는 기억기배정요구를 만족시키기에는 너무나 작은 블로크들로 인차 분산되는것으로 된다. 따라서 다른 알고리듬들에 비하여 기억기조이기가 더 자주 진행되여야 한다.

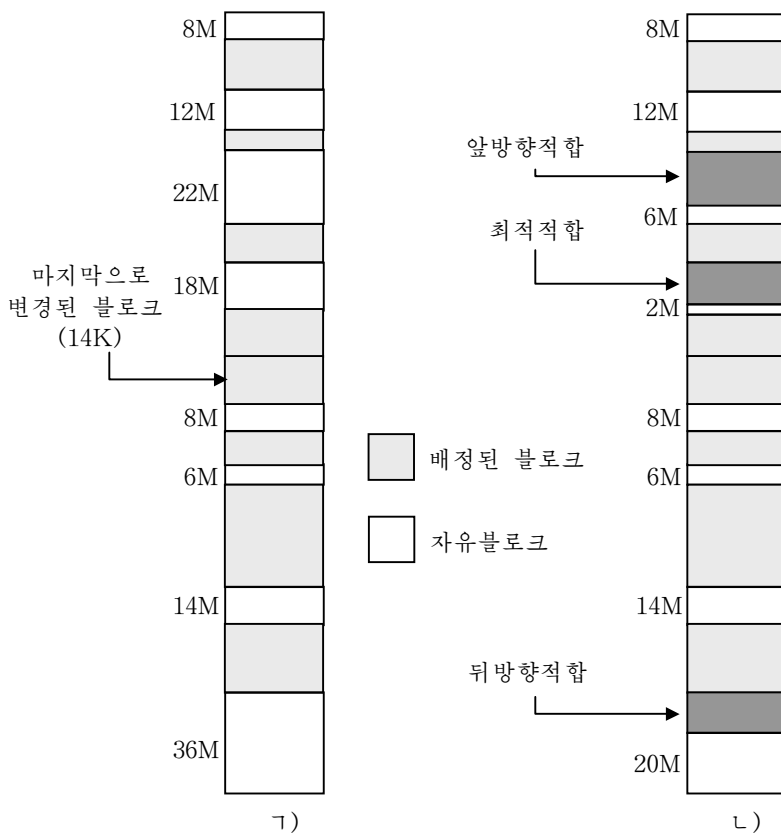



그림 7-5. 16MB블로크의 배정전후의 기억기구성실례
ㄱ-전, ㄴ-후

치환알고리듬

동적분할을 쓰는 다중프로그람처리체계에서는 주기억기안에 있는 프로쎄스들이 모두 블로크화된 상태에 있고 조이기를 한후에도 프로쎄스를 추가하기에는 기억기가 모자라는 경우가 있다. 능동프로쎄스가 비페색상태로 되기를 기다리는 처리기시간의 랑비를 없애기 위해서는 조작체계가 주기억기안에 있는 프로쎄스들중의 한개를 새로운 프로쎄스나 준비-중단상태에 있는 프로쎄스를 위한 공간을 확보하기 위하여 교체하여 주기억기밖으로 내 보내야 한다. 따라서 조작체계는 어느 프로쎄스를 교체할것인가를 선택하여야 한다. 교체알고리듬이 여러가지 가상기억방안들에 대한 몇가지 세부를 포괄하기때문에 교체알고리듬에 대한 해설을 그때에 가서 하기로 한다.

동료체계

고정분할방법과 동적분할방법은 둘다 결함을 가지고 있다. 고정분할방법은 능동프로쎄스의 수를 제한하며 사용가능한 분할구역의 크기와 프로쎄스의 크기가 잘 맞지 않으면 기억공간을 비효률적으로 사용할수 있다. 동적분할방법은 관리가 보다 복잡하며 조이기를 하는데 추가적인 시간이 든다. 흥미 있는 타협안이 바로 동료체계이다([KNUT 97], [PETE77]).

동료체계에서 기억기블로크들은 2^k , $L \leq K \leq U$의 크기를 가질수 있다. 여기서

2^L = 배정되는 가장 작은 블로크의 크기
2^U = 배정되는 가장 큰 블로크크기; 일반적으로는 2^U가 배정에 사용가능한 전체 기억기의 크기로 된다.

배정을 시작할 때 사용가능한 전체 기억공간은 2^U의 크기를 가진 하나의 블로크를 취급한다. $2^{U-1} < s \leq 2^U$ 만한 크기 s의 요구가 제기되면 옹근블로크가 배당된다. 그렇지 않으면 블로크는 2^{U-1}만한 크기의 똑같은 두개의 동료들로 갈라 진다. 만일 $2^{U-2} < s \leq 2^{U-1}$이면 요구는 두개 동료들중의 하나에 배정된다. 그렇지 않으면 동료들중의 하나는 다시 절반으로 갈라진다. 이러한 과정은 s보다 크거나 같은 가장 작은 블로크가 생겨 나고 요구에 따라 그것이 배정될 때까지 계속된다. 임의의 시각에 동료체계는 매개의 크기가 2^i인 구멍(배정되지 않은 블로크)들에 대한 목록을 유지하고 있다. i번째 목록에 있는 크기가 2^i인 두개의 동료를 창조하기 위하여 (i+1)번째 목록을 둘로 가르는것으로 구멍을 제거할수 있다. i번째 목록에서 두개의 동료가 다 배정되지 않으면 그것들은 목록에서 삭제되고 (i+1)번째 목록의 한개의 블로크로 합쳐 진다. $2^{i-1} < s \leq 2^i$인 k크기를 배정하는 요구가 제기되면 다음의 회귀알고리듬([LIST 93])은 크기가 2^i인 구멍을 찾는데 사용한다. 즉

```
void get_hole ( int  i )
{
          if ( i  = =  (U + 1) )
             <failure>;
          if ( < i_list empty > )
          {
          get_hole  ( i+1 ) ;
          < split  hole  into  buddies > ;
          < put  buddies  on  i_list > ;
          }
          < fake  first  hole  on  i_list > ;
}
```

그림 7-6에서는 1MB의 초기블로크를 사용하는 실례를 보여 주고 있다. 첫번째 요구 A는 100KB를 배정하는것이므로 128K블로크가 필요하다. 이때 초기블로크는 두개의 512K 동료로 나누어 진다. 이것들중의 첫번째의것이 두개의 256K쌍으로 갈라 지고 또 이것들중의 첫번째의것이 두개의 128K로 갈라 지며 그것들중의 하나가 A에 배정된다. 다음의 요구 B에는 256K블로크가 필요하다. 그러한 블로크는 이미 사용할수 있으므로 즉시 배정된다. 이러한 과정은 필요할 때까지 분할과 통합을 계속한다. E가 해방되였을 때 두개의 128K쌍은 256K블로크로 통합되고 또 그것들은 즉시에 자기들의 동료와 통합된다.

그림 7-7에서는 B요구가 해방된 직후의 동료배정의 2진나무표현을 보여 주고 있다. 잎마디들은 기억기의 현재 분할을 나타 낸다. 만약 두 동료가 잎마디라면 최소한 하나는 배정되여야 하며 그렇지 않으면 보다 큰 블로크로 통합되여야 한다.

동료체계가 고정분할과 동적분할의 두가지 방안의 결함을 극복하기 위한 합리적인 타협안이기는 하지만 현대적인 조작체계에서는 페지화와 토막화에 기초한 가상기억기가 보다 우월하다. 그러나 동료체계는 병렬체계에서 병렬프로그람들을 배치하고 개발하는 효과적인 수단으로서 사용되고 있다([JOHN 92]를 보라.). 동료체계의 개선된 형태가 UNIX의 핵심부의 기억기배정에서 사용되고 있다(이것에 대해서는 제8장에서 서술하였다.).

1Mbyte 요청	1M					
100K 요청	A=128K	128K		B=256K	512K	
240K 요청	A=128K	128K		B=256K	512K	
64K 요청	A=128K	C=64K	64K	B=256K	512K	
256K 요청	A=128K	C=64K	64K	B=256K	D=256K	256K
B 해방	A=128K	C=64K	64K	256K	D=256K	256K
A 해방	128K	C=64K	64K	256K	D=256K	256K
75K 요청	E=128K	C=64K	64K	256K	D=256K	256K
C 해방	E=128K	128K		256K	D=256K	256K
E 해방	512K				D=256K	256K
D 해방	1M					

그림 7-6. 동료체계의 실례 1M

재배정

분할법의 부족점을 퇴치하는 방법을 고찰하기전에 기억기상에서 프로쎄스의 배치와 관련되는 한가지 애매한 문제를 명백히 하여야 한다. 그림 7-3 ㄱ의 고정분할방법을 사용하는 경우에는 프로쎄스가 언제나 같은 분할구역에 할당될것이라고 할수 있다. 다시 말하여 이 방법은 새로운 프로쎄스를 적재할 때 어느 분할구역이 선택되든지간에 그것이 교체되여 나간후에 프로쎄스를 기억기에로 도로 교체하여 넣는데 사용된다. 그러한 경우에는 부록 7-ㄱ에서 서술한것과 같은 단순재배정적재기를 쓸수 있다. 프로쎄스가 처음 적재될 때 코드안에 있는 모든 상대적인 기억기참조들은 적재된 프로쎄스의 기준주소로 결정되는 주기억기절대주소에 의하여 바뀌여 진다.

분할구역들의 크기가 같은 경우(그림 7-2)에 또한 크기가 같지 않은 분할구역들에 대한 한개의 프로쎄스대기렬의 경우(그림 7-3ㄴ)에 프로쎄스는 그것의 생명주기들만 서로다른 분할구역들을 차지할수 있다. 프로쎄스형태가 처음 창조되였을 때 그것은 주기억기의 몇개의 분할구역들에 적재된다. 후에 프로쎄스는 교체되여 나갈수 있다. 후에 그것이 도로 교체되여 들어올 때는 다른 분할구역에 할당될수 있다. 동적분할인 경우에도 사정은 마찬가지이다. 그림 7-4 ㄷ와 7-4 ㅇ에서는 프로쎄스 2가 끌려 들어 오는 두가지 경우에 주기억기의 서로다른 두 구역을 차지하는것을 보여 주고 있다. 한편 조이기를 할 때 프로쎄스들은 주기억기안에서 이동한다. 이렇게 프로쎄스가 참조하는 위치(명령과 자료의 위치)는 고정되여 있지 않다. 이것은 프로쎄스가 교체되여 들어 오거나 이동할 때마다 매번 변한다. 이 문제를 해결하기 위하여 서로 구별되는 몇가지 형태의 주소지정방법을 사용한다. **론리주소**는 현재의 기억기자료할당과 무관계한 기억기위치참조이다. 이때에는 기억기에 접근하기전에 그것을 물리주소로 변환하여야 한다. **상대주소**는 론리주소의 특수한 실례인데 여기서는 주소가 보통 프로그람의 시작점과 같은 몇개의 알려진 점에 대한 상대적인 위치로 표현된다. **물리주소** 또는 절대주소는 주기억기안의 실제적인 위치이다.

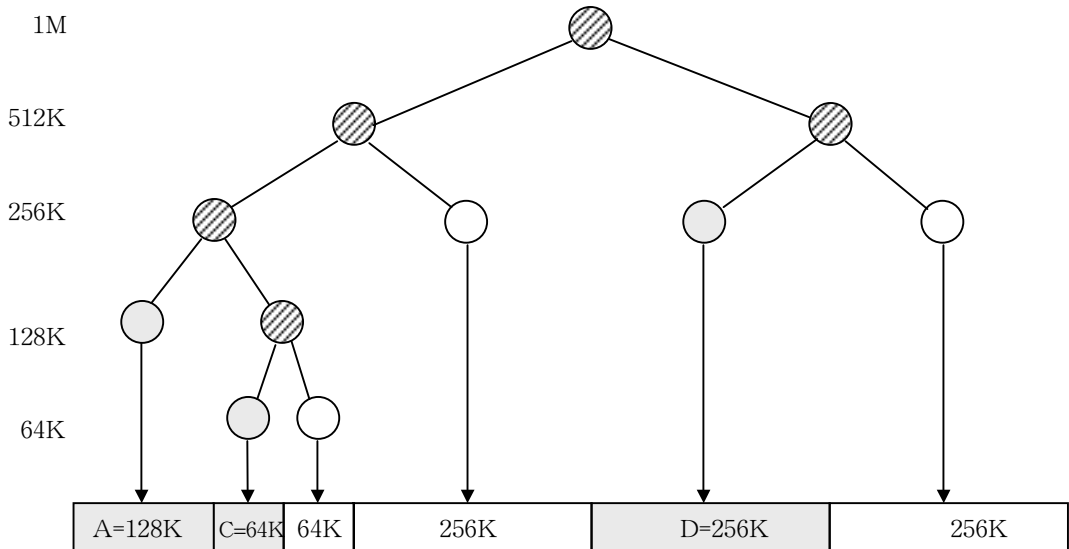



그림 7-7. 동료체계의 나무표현

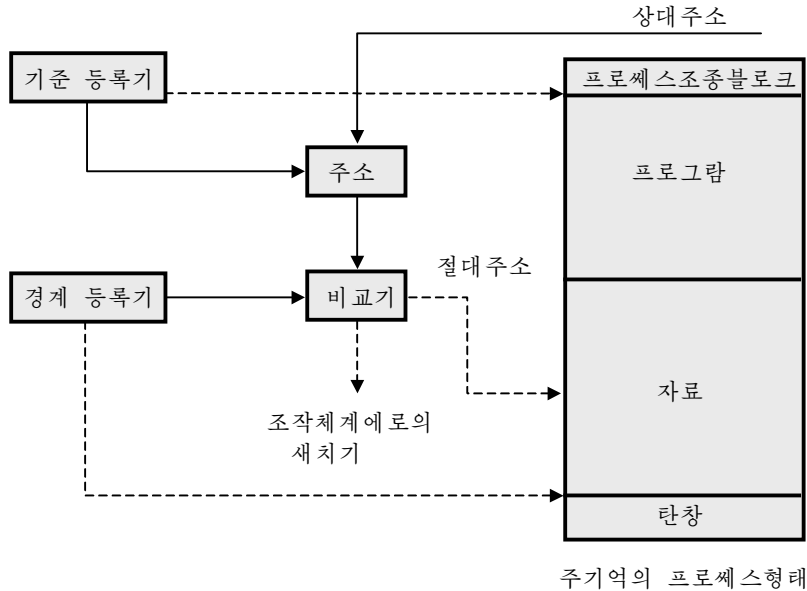



그림 7-8. 재배정에 대한 하드웨어적지원

기억기상대주소를 쓰는 프로그람들은 동적실행시적재방법을 사용하여 적재된다(상세한 것은 부록 7-ㄱ을 볼것). 이것은 적재된 프로쎄스안의 모든 기억기참조값들이 프로그람의 원점에 대하여 상대적이라는것을 의미한다. 그리하여 참조를 포함하는 명령을 실행할 때에는 상대주소를 주기억기의 물리주소로 변환하는 하드웨어적인 기구가 필요하다

그림 7-8에서 주소변환을 대표적으로 진행하는 방법을 보여 주고 있다. 처리기의 전용등록기에 주기억기의 프로쎄스시작주소가 넣어 진다. 프로그람의 끝위치를 지정하는 경계등록기도 있다. 이 값(끝위치값)들은 프로그람이 기억기에 적재되거나 프로쎄스가 교체넣기될 때 설정되여야 한다. 프로쎄스의 실행과정에 상대주소들과 맞다 들게 된다. 이것들은 명령등록기의 내용들, 갈래명령과 호출명령들에서 나타나는 명령주소들, 적재명령과 기억명령들에서 나타나는 자료주소들을 포함한다. 그러한 매개 상대주소들은 처리기가 진행하는 두 단계의 조작을 통과하여 나간다. 우선 절대주소를 만들기 위하여 기준등록기값을 상대주소에 더한다. 다음으로 얻어진 주소값을 경계등록기의 값과 비교한다. 주소값이 경계안에 있으면 명령실행은 전진한다. 그렇지 않으면 여러가지 방법으로 오유에 대응하는 새치기가 조작체계에 발생된다.

그림 7-8에 제시한 방안에서는 실행과정에 프로그람들이 기억기의 안팎으로 교체될 수 있다. 그것은 또한 일정한 정도의 보호기능을 준다. 즉 매개 프로쎄스형태는 기준 및 경계등록기의 내용에 의하여 서로 분리되므로 다른 프로쎄스가 불필요한 접근을 하는데 대해서는 안전하다.

제 3 절. 페지화

동일하지 않은 고정크기나 가변크기를 가진 분할구역들은 다 같이 기억기사용에서 효률적이지 못하다. 전자는 내부쪼각들을 만들어 내며 후자는 외부쪼각들을 만들어 낸다. 한편 주기억기가 상대적으로 작은 동일한 고정크기의 토막들로 분할되고 매개 프로쎄스가 또한 같은 크기의 작은 고정크기토막들로 나뉘여 진다고 가정하자. 그러면 **페지**라고

하는 프로쎄스토막들은 **프레임** 혹은 페지프레임이라고 부르는 기억기의 사용가능한 토막들에 할당할수 있다. 프레임이라는 술어는 프레임이 한개의 자료페지를 담거나 틀에 맞출수 있기때문에 사용한다. 이 절에서는 매개 프로쎄스에 의하여 랑비되는 기억공간이 프로쎄스의 마지막 페지의 한 단편으로만 이루어 지는 내부쪼각탓이라는것을 보게 된다. 외부쪼각은 없다.

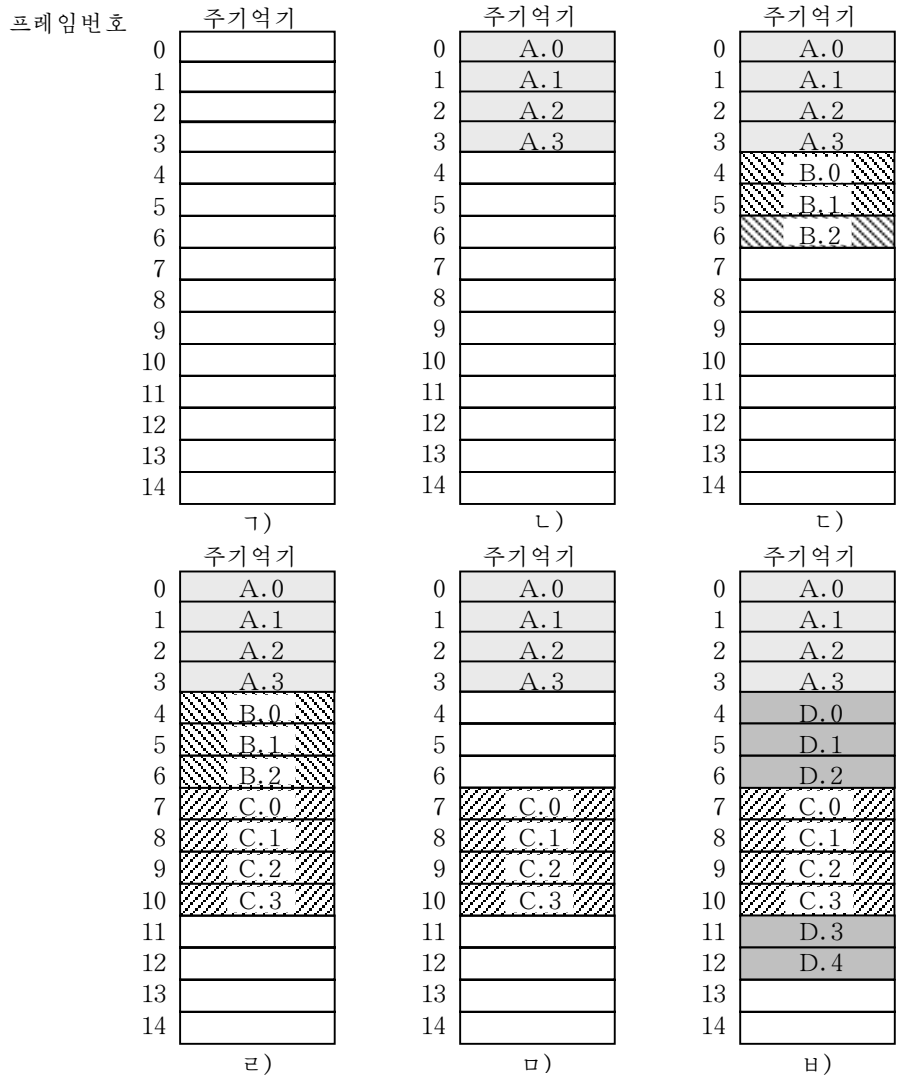



그림 7-9. 자유프레임에 프로쎄스의 할당
ㄱ-15개의 사용가능한 프레임, ㄴ-프로쎄스 A의 적재, ㄷ-프로쎄스 B의 적재
ㄹ-프로쎄스 C의 적재, ㅁ-B의 교체내기, ㅂ-프로쎄스 D의 적재

그림 7-9에서 페지와 프레임들의 사용실례로 보여 주고 있다. 일부는 자유로운 상태에 있다. 조작체계는 자유프레임들의 목록을 관리한다. 디스크에 넣어 진 프로쎄스 A는 네개의 페지를 이룬다. 이 프로쎄스를 적재할 때가 되면 조작체계는 네개의 프레임에 적재한다(그림 7-9 ㄴ). 세개의 페지로 이루어 지는 프로쎄스 B와 네개의 페지로 이루어 지는 프로쎄스C도 뒤따라 적재된다. 다음에 프로쎄스 B가 중지되고 주기억기밖으로 교체되여 나간다. 그뒤에 주기억기안에 있는 모든 프로쎄스들이 페색되며 조작체계는 다섯개의 페지로 이루어 지는 새로운 프로쎄스인 프로쎄스 D를 끌어 들인다.

이제 이 실례에서와 같이 프로쎄스를 담을만큼 충분하고 사용하지 않는 련속된 프레임들이 없다고 가정하자. 이것이 조작체계가 프로쎄스D를 적재할수 없게 하겠는가? 대답은 《아니》이다. 그것은 론리주소의 개념을 다시 한번 사용할수 있기때문이다. 단순한 기준주소등록기가 더이상 충분하지 않다. 오히려 조작체계는 매 프로쎄스에 대한 **페지표**를 관리한다. 페지표는 프로쎄스의 매개 페지에 대한 프레임의 배치를 보여 준다. 프로그람안에서 매개 론리주소들은 페지번호와 페지안에서의 편위로 구성된다. 단순분할인 경우에 론리주소가 프로그람의 시작과 관계되는 단어의 위치이고 처리기는 그것을 물리주소로 변환한다는것을 상기하자. 페지화에서 론리주소-물리주소변환은 여전히 처리기하드웨어에 의해 진행된다. 이때 처리기는 현행프로쎄스의 페지표를 어떻게 호출하겠는가를 알아야 한다. 론리주소(페지번호와 편위)로 제시된것을 처리기는 물리주소(프레임번호와 편위)로 얻어 내기 위해 페지표를 사용한다.

우에서 한 설명을 계속하면 프로쎄스 D의 5개의 페지들은 프레임 4, 5, 6, 11, 12에 적재한다. 그림 7-10에서는 이 시점에서 여러가지 페지표를 보여 주고 있다. 페지표는 프로쎄스의 매개 페지에 한개의 입구점을 포함하고 있어서 표는 페지번호(0페지로부터 시작하여)에 의해 쉽게 색인할수 있다. 매개 페지표입구점들은 주기억기의 프레임번호를 보유하고 있으며 그것은 대응하는 페지를 보관한다. 또한 조작체계는 현재 차지하지 않고 페지들에 대하여 사용가능한 주기억기의 모든 프레임들의 한개의 자유프레임목록을 유지한다.

이상에서 고정분할법과 류사한 단순페지화를 보았다. 고정분할법과의 차이점은 페지화에서는 할당구역이 보다 작고 프로그람이 한개 상의 할당구역들을 차지할수 있으며 이 할당구역들이 련속되여야 할 필요가 없다는것이다.

페지화방안은 사용하기 편리하게 하기 위하여 페지크기와 프레임크기를 2의 제곱으로 하자. 페지크기를 2의 제곱으로 하여 사용하면 프로그람의 편위로 표시되는 론리주소가 동일하다는것을 쉽게 알수 있다. 그 실례를 그림 7-11에서 보여 주고 있다. 이 실례에서는 16bit주소를 사용하고 페지크기는 1K=1024byte이다. 상대주소 1502는 2진표현으로서 0000010111011110이다. 페지크기가 1K 인 경우에는 10bit의 편위마당이 필요하므로 6bit를 페지번호용으로 남겨 놓는다. 이렇게 하여 프로그람은 매개가 1Kbyte인 최대 =64개의

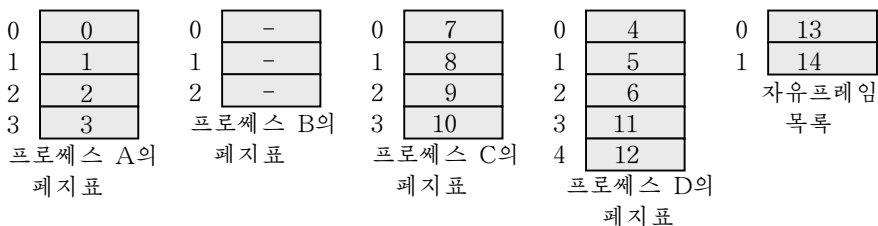



그림 7-10. 시간훈련패턴렬에서 그림 7-9 의 실례에 대한 자료구조

페지들로 구성할수 있다. 그림 7-11에서 보여 준바와 같이 상대주소 1502는 페지 1(000001)의 편위 478(0111011110)에 대응하는데 이것은 같은16bit수인 00000101 11011110을 낳게 된다.

2의 제곱인 페지크기를 사용하면 두가지 결과가 얻어 진다. 첫째로, 론리주소화방안이 프로그람작성자, 아쎔블러 및 련결기에게 투명한것이다. 프로그람의 매개 론리주소(페지번호와 편위)는 그것의 상대주소와 꼭 같다. 둘째로, 실행시에 동적으로 주소변환을 진행하는 하드웨어의 기능을 실현하기가 상대적으로 쉽다. $n + m$ bit의 주소를 생각하자. 여기서 왼쪽의 n bit는 페지번호이고 오른쪽의 m bit는 편위이다. 실례에서(그림 7-11 ㄴ) $n = 6$이고 $m = 10$이다. 주소를 변환하기 위해서 다음의 단계를 거쳐야 한다. 즉

- 론리주소의 왼쪽 n bit로서 페지번호를 따 낸다.
- 프레임번호 k를 찾기 위하여 프로쎄스페지표에 대한 색인으로서 페지번호를 사용한다.
- 프레임의 시작물리주소는 $k \times 2^m$이고 참조하는 바이트의 물리주소는 페지번호에 편위를 더한 값이다. 이 물리주소는 계산할 필요가 없다. 그것은 편위에 프레임번호를 붙여서 쉽게 얻어 진다.

실례에서는 론리주소 0000010111011110을 사용하는데 여기서 페지번호는 1이고 편차는 478이다. 이 페지가 주기억안의 프레임 6 (2진수로 00010)에 있다고 가정하자. 그러면 물리주소는 프레임번호 6과 편위 478=0001100111011110 (그림 7-12 ㄱ)이다.

요약하면 단순페지화에서는 주기억기를 많은 같은 크기의 작은 프레임들로 나눈다.

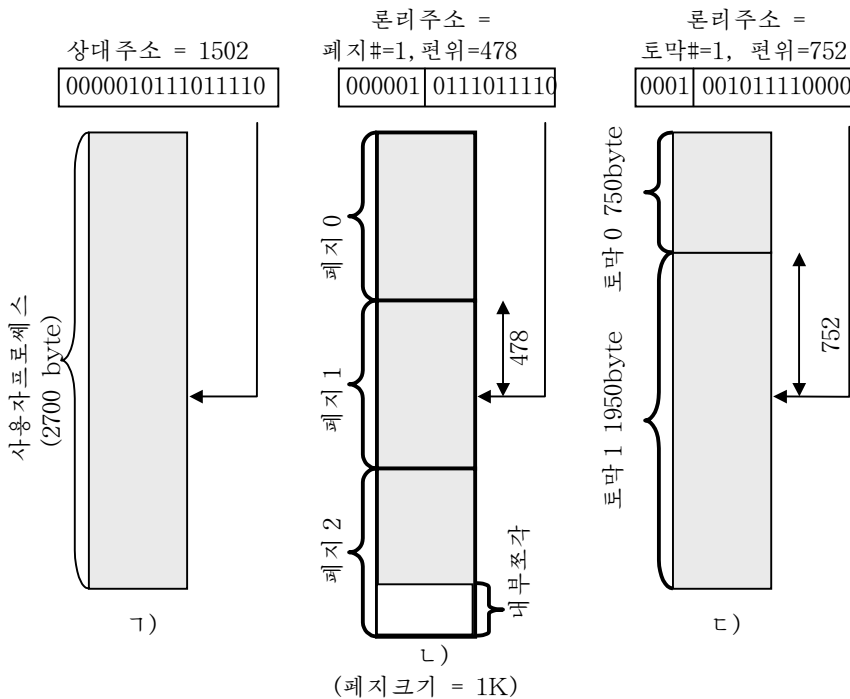



그림 7-11. 론리주소
ㄱ-분할법, ㄴ-페지화, ㄷ-토막화

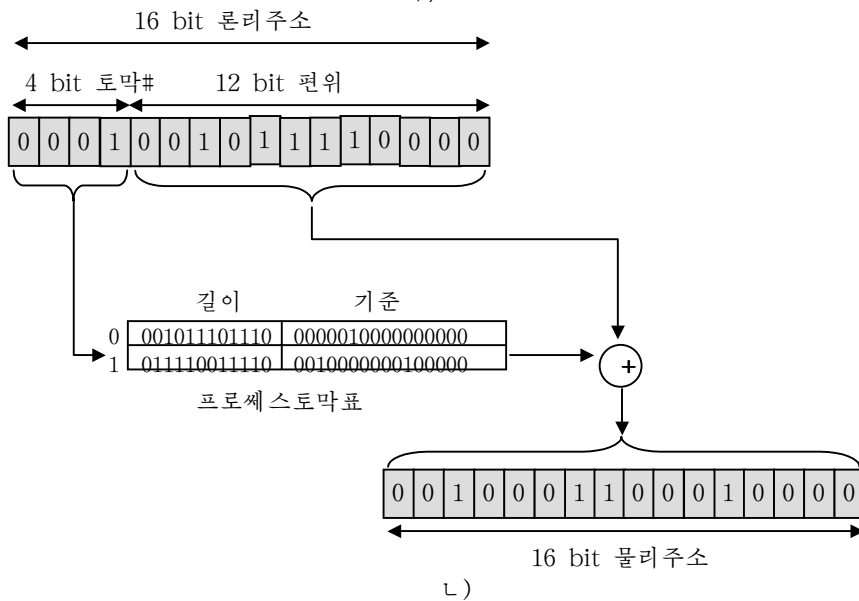
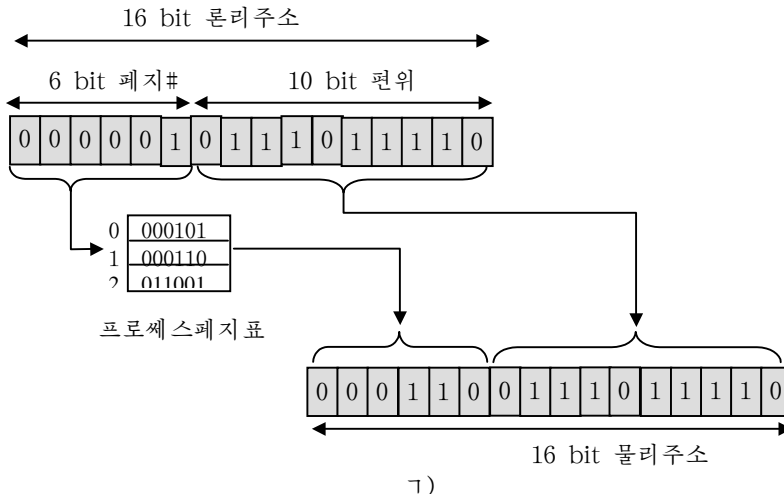



그림 7-12. 론리물리주소변환의 실례
ㄱ-페지화, ㄴ-토막화

매개 프로쎄스는 프레임크기의 페지들로 나눈다. 보다 작은 프로쎄스들은 불과 몇개의 페지를 요구할것이고 보다 큰것은 더 많은 페지를 요구할것이다. 프로쎄스를 끌어 들일때 그것의 모든 페지들은 사용가능한 프레임들에 적재되며 페지표가 작성된다. 이러한 방법은 분할에 고유한 여러가지 문제를 해결한다.

제 4 절. 토막화

사용자프로그람을 세분할수 있는 다른 방법은 토막화이다. 이 경우에 프로그람과 그것과 관련되는 자료는 여러개의 **토막**으로 갈라 진다. 모든 프로그람의 토막들이 같은 길이를 가질것은 요구하지 않으며 지어 최대토막길이를 가질수 있다. 페지화에서와 같이

토막화가 사용하는 론리주소는 두 부분으로 이루어 지는데 이 경우에 하나는 토막번호이고 하나는 편위이다.

크기가 같지 않은 토막을 사용하므로 토막화는 동적분할법과 류사하다. 겹쳐놓기방안이나 가상기억기를 사용하지 않는 상태에서 프로그람의 모든 토막들이 집행을 위해 기억기에 적재되여야 한다. 동적분할과 비교해 볼 때 차이점은 토막화에서는 프로그람이 한개이상의 분할구역을 차지할수 있고 이 분할구역들은 련속되여 있을 필요가 없는것이다. 토막화는 동적분할법과 같이 내부쪼각은 제거하지만 외부쪼각은 그대로 생긴다. 그러나 프로쎄스가 여러개의 보다 작은 파편들로 갈라 지기때문에 외부쪼각은 작아 진다.

페지화가 프로그람작성자에게는 보이지 않지만 반대로 토막화는 일반적으로 보이며 프로그람과 자료를 조직하는데 편리한 수단을 준다. 대표적으로 프로그람작성자나 콤파일러는 프로그람과 자료를 다른 토막들에 할당한다. 모듈식프로그람작성에서 프로그람이나 자료는 여러개의 토막들로 보다 더 갈라 질수 있다. 이러한 봉사가 원리적으로 불편한 점은 프로그람작성자가 토막의 최대크기한계를 알고 있어야 하는것이다.

비동일크기토막들의 다른 결함은 론리주소와 물리주소사이의 관계가 단순하지 않은것이다. 페지화와 류사하게 단순토막화방안은 매개 프로쎄스에 대한 토막표와 주기억기의 자유블로크목록을 사용한다. 매개 토막표입구점은 주기억기안에서 대응하는 토막의 시작주소를 주어야 한다. 입구점은 또한 무효한 주소가 사용되지 않도록 담보하기 위하여 토막길이를 주어야 한다. 프로쎄스가 실행상태에 들어 갈 때 그것의 토막표주소는 기억기관리하드웨어가 사용하는 전용등록기에 적재된다. $n+m$ bit의 주소를 생각하자. 여기서 왼쪽 n bit는 토막번호이고 오른쪽 m bit는 편위이다. 실례 (그림 7-11 ㄷ)에서는 $n=4$ 이고 $m=12$이다. 이렇게 하여 최대토막크기는 4096이다. 주소변환을 위하여 다음의 단계를 거쳐야 한다. 즉

- 론리주소의 왼쪽 n bit로서 토막번호를 따낸다.
- 토막의 시작물리주소를 찾기 위하여 토막번호를 프로쎄스토막표에 대한 첨수로 사용한다.
- 오른쪽 m bit로 표시된 편위를 토막길이와 비교한다. 편위가 길이보다 더 크다면 주소는 무효하다.
- 토막의 시작물리주소와 편위와의 합이 요구하는 물리주소이다.

실례에서 론리주소 0001001011110000을 사용하는데 여기서 토막번호는 1, 편위는 752이다. 이 토막이 물리주소 0010000000100000에서 시작하는 주기억기에 있다고 가정하자. 그러면 물리주소는 0010000000100000 + 001011110000 = 0010001100010000이다(그림 7-12 ㄴ)

요약하면 단순토막화에서 프로쎄스는 크기가 같지 않아도 되는 여러개의 토막들로 나누어 진다. 프로쎄스를 끌어 들여 올 때 그것의 모든 토막들은 기억기의 사용가능한 령역에 적재되며 토막표가 작성된다.

요약, 기본용어 및 복습문제

조작체계의 가장 중요하고도 복잡한 과제의 하나는 기억기관리이다.기억기관리는 여러개의 능동프로쎄스들을 배정하고 공유하기 위한 자원으로서의 주기억기를 취급한다.처리기와 입출력설비들을 효률적으로 사용하기 위해서는 될수록 주기억기안에 많은 프로쎄스를 유지하는것이 좋다. 더우기 프로그람개발에서 프로그람작성자들은 크기의 제약을 받지 않도록 하는것이 좋다.

$$Ni = Ai + Gi + Li$$

일반적으로 지연동료체계는 국부자유블로크들의 집결소를 유지하려고 하며 다만 국부자유블로크들의 수가 턱값을 넘으면 합치기를 개시한다. 국부자유블로크가 너무 많다면 요구를 만족시켜야 할 다음 준위에서 자유블로크들이 모자랄수 있는 기회가 생긴다. 블로크가 자유로워 질 때 대부분의 시간은 합치기를 발생하지 않으며 그래서 최소의 예산과 조작비용이 든다. 블로크가 배정될 때 국부적으로 및 전역적으로 자유로운 블로크들사이에서 명백한 차이가 이루어 지지 않으며 또 이것이 예산을 최소화한다.

합치기에 사용된 기준은 주어 진 크기의 국부자유블로크들이 그 크기의 배정된 블로크의 수를 넘지 말아야 한다는것이다(즉 $Li \leq Ai$ 가 만족되여야 한다.). 이것은 국부자유블로크들의 확대를 제한하는 적당한 안내방향인데 [BARK89]에서의 실험은 이 방안이 현저한 절약을 가져 온다는것을 확증하고 있다.

이 방안을 실현하기 위해 다음과 같이 지연변수를 정의한다. 즉

$$Di = Ai - Li = Ni - 2Li - Gi$$

그림 8-24에서는 그 알고리듬을 보여 주고 있다.

Di의 초기값은 0이다.
조작후에 Di의 값은 다음과 같이 변경된다.

(I) 만일 다음조작이 블로크배정요청이면:
만일 자유블로크가 있으면 배정하기 위해 한개를 설정한다.
만일 선택된 블로크가 국부적으로 자유로우면
그러면 Di:=Di+2
아니면 Di:=Di+1
한편
우선 보다 큰 한개를 두개로 쪼개여 두개의 블로크를 만든다(재귀조작).
한개를 배정하고 다른 국부적으로 자유로운것에 표식을 단다.
D_i는 변화시키지 않은채로 남겨 둔다(그러나 D_i는 재귀호출로 하여 다른 블로크크기로 변화시킬 수 있다.).

(II) 만일 다음 조작이 블로크자유요청이면
Di≧2인 경우
그것에 국부자유표식을 하고 그것을 국부적으로 자유롭게 한다.
Di:=Di-2
Di=1인 경우
그것에 전역자유표식을 하고 그것을 전역으로 자유롭게 한다; 가능하면 합친다.
Di:=0
Di=0인 경우
그것에 전역자유표식을 하고 그것을 전역으로 자유롭게 한다; 가능하면 합친다.
크기가 2^i인 국부적으로 자유로운 한개의 블로크를 선택하고 그것을 전역으로 자유롭게 한다; 가능하면 합친다.
Di:=0

그림 8-24. 지연동료체계의 알고리듬

제 4 절. Linux의 기억기관리

LINUX는 다른 UNIX실현물들의 기억기관리방안중에서 많은 기능들을 공유하고 있으면서도 자기자체의 통일적인 특징을 가지고 있다. 총적으로 LINUX의 기억기관리방안

련 습 문 제

1. 제 2 장 제 3 절에서 기억기관리와 5 가지 대상을 렬거하였고 제 7 장 제 1 절에서는 5 가지 요구사항을 들었다. 매개 항목들이 다른 항목들에서 지적한 모든 관계들을 포괄한다는것을 증명하시오.

2. 동적분할방안을 생각하자. 여기서 기억기는 평균적으로 토막들의 절반만한 구멍들을 포함한다는것을 증명하시오.

3. 동적분할법을 취급（제7장 제2절）한 여러가지 배치알고리듬들을 실현하기 위해서는 기억기의 자유블로크들의 목록을 가지고 있어야 한다. 해설한 세가지 방법들（최적적합, 웃방향적합, 아래방향적합）중 매개 방법에서 탐색의 평균길이는 얼마인가?

4. 기억기의 제일 큰 자유블로크가 사용된다. 웃방향적합, 아래방향적합, 최적적합방법들과 비교하여 이 방법의 우결함을 분석하시오. 최대비적합법의 평균탐색길이는 얼마인가?

5. 동료체계를 써서 1Mbyte의 기억기블로크 A를 배치하였다.

   ㄱ) 다음의 순서렬의 결과를 그림 7-6과 류사한 그림으로 표시하시오. 70K요구, 35K요구, 80K요구； A복귀, 60K요구； B복귀； D복귀； C복귀

   ㄴ) B복귀다음의 2진나무표현을 그리라.

6. 동료체계에서 현재 배정된 특정한 블로크의 주소가 011011110000이라고 하자

   ㄱ) 블로크의 크기가 4이면 그 동료의 2진주소는 얼마인가？

   ㄴ) 블로크의 크기가 16이면 그 동료의 2진주소는 얼마인가？

7. 주소가 x이고 크기가 2^k인 블로크의 동료주소가 $\mathrm{Buddy}_k(x)$라고 하자. $\mathrm{Buddy}_k(x)$의 일반식을 쓰시오.

8. 피보나치렬을 다음과 같이 정의한다.

$$F_0 = 0,\ F_1 = 1, \qquad F_{n+2} = F_{n+1} + 1 + F_n, \quad n>=0$$

   ㄱ) 이 수렬을 동료체계로 확립하는데 사용할수 있는가?

   ㄴ) 이 장에서 서술한 2진동료체계에 비해서 이 체계의 우점은 무엇인가?

9. 프로그람실행과정에 처리기는 명령등록기(프로그람계수기)의 내용을 매번 명령불러내기한 다음에 한 단어씩 증가시킨다. 그러나 만약 등록기가 프로그람안의 다른 곳에서 실행을 계속하게 하는 갈래명령이나 호출명령과 맞다 들게 되면 등록기의 내용이 바뀌여 진다. 이제 그림 7-8을 생각하자. 명령주소에 관해서는 두가지 방도가 있다.

   - 명령등록기는 상대주소가 가지고 있고 입력으로서 명령등록기를 사용하여 동적주소변환을 진행한다. 성공적인 갈래명령이나 호출명령과 맞다 들었을 때 갈래명령과 호출명령에 의해 생기는 상대주소는 명령등록기에 적재된다.
   - 명령등록기는 절대주소를 가지고 있다. 성공적인 갈래명령이나 호출명령과 맞다들었을 때 명령등록기에 보관된 결과를 가지고 동적주소변환을 사용한다.

   어느 부름법이 더 우월한가?

10. 페지화체계에서 가상주소 a는 수쌍 $(p,\ w)$와 등가이다. 여기서 p는 페지번호이고 w는 페지안의 바이트번호이다. z를 페지안의 바이트들의 개수라고 하자. p와

w를 z와 a의 함수로 보여 주는 대수방정식을 작성하라.

11. 다음의 그림에서 보여 준바와 같이 기억기의 한 끝으로부터 다른 끝으로 만들어 진 순서대로 토막 S_1, S_2, … S_n들이 린접하여 놓여 진 기억기를 생각하자.

S_1	S_2	…	S_n	구멍

토막 S_{n+1}이 만들어 질 때 그것은 비록 토막 S_1, S_2, … S_n중에서 일부가 이미 삭제될수 있다고 해도 토막 S_n의 바로 뒤에 배치된다. 토막들(사용중이거나 삭제된 것)과 구멍사이의 경계가 기억기의 다른 끝에 도달할 때 사용중에 있는 토막들은 조여 진다.

ㄱ) 조이기에 소비하는 시간토막 F가 다음의 부등식에 따른다는것을 증명하라.

$$F \ge \frac{1-f}{1+kf} \qquad \text{여기서} \quad k = \frac{t}{2s} - 1$$

여기서 s = 단어로 표시한 토막의 평균길이

t = 기억기참조에서 토막의 평균수명

f = 균형상태에서 사용되지 않는 기억기파편

주의: 경계가 기억기를 횡단하는 평균속도를 찾아 내고 한 단어의 복사가 최소한 두번의 기억기참조를 요구한다고 가정하라.

ㄴ) f = 0.2, t = 1000, s = 50에 대하여 F를 계산하여라.

부록 7-ㄱ. 적재와 련결

능동프로쎄스창조의 첫 단계는 프로그람을 주기억기에 적재하고 프로쎄스형태를 만드는것이다(그림 7-13). 그림 7-14는 대부분의 체계들에서 대표적인 구성안을 제시하였다. 응용프로그람은 목적코드형식으로 콤파일 또는 아쎔블리된 여러개의 모듈들로 이루어 졌다. 이 모듈들은 그것들사이의 참조를 위하여 서로 련결된다. 동시에 서고루틴들을 참조한다. 서고루틴들 그자체는 프로그람안에 통합되거나 또는 실행시에 조작체계가 지원하여야 하는 공유된 코드로서 창조될수 있다. 이 부록에서는 련결기와 적재기의 기본기능에 대하여 요약해서 설명한다. 리해를 명확히 하기 위하여 한개의 프로그람모듈만을 포함하여 련결이 요구되지 않을 때의 적재과제를 해설하는것으로부터 시작한다.

적재

그림 7-14에서 적재기는 위치 x에서 시작하여 주기억기에 적재모듈을 배치한다. 프로그람은 적재할 때 그림 7-1에서 설명한 주소지정의 요구사항을 만족해야 한다. 일반적으로 세가지 방법을 적용할수 있다. 즉

- 절대적재
- 재배정가능한 적재
- 동적실행시적재

절대적재

절대적재기는 주어 진 적재모듈이 항상 주기억기의 같은 위치에 적재될것을 요구한다. 이로부터 적재기에 표현된 적재모듈에서 모든 주소참조들은 특정한 혹은 절대적인 주기억주소가 되여야 한다. 실례로 그림 7-14에서 x가 위치 1024이면 기억기에 그 령역을 사용하기로 예정된 적재모듈의 첫번째 단어는 주소 1024를 가진다.

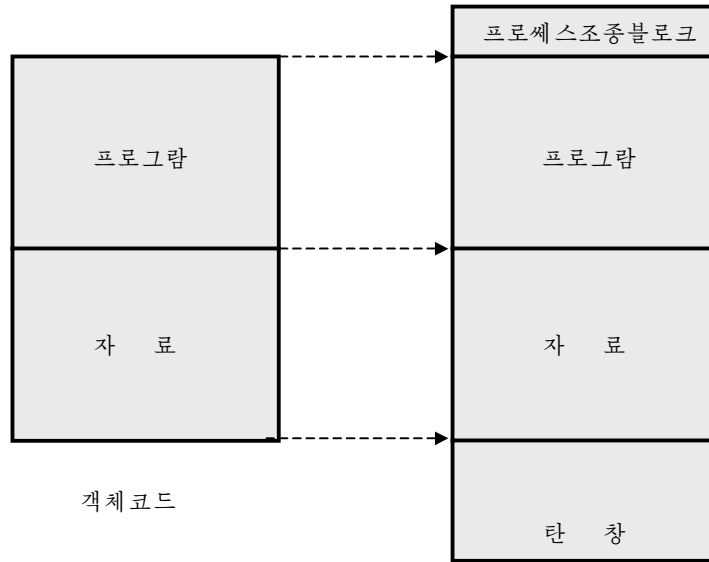



그림 7-13. 적재기능

프로그람안의 기억기참조들에 명시한 주소값을 할당하는것은 프로그람작성자가 하거나 콤파일 또는 아쎔블리할 때 할수 있다(표 7-2 ㄱ). 앞의 방법에는 여러가지 불리한 점이 있다. 그것은 첫째로, 매개 프로그람작성자가 주기억기에 모듈들을 배치하기 위하여 계획한 할당전략을 알고 있어야 한다. 둘째로, 모듈본체에 대한 삽입이나 삭제를 위하여 프로그람에 어떤 변경이 가해 져도 모든 주소들이 바꾸어 지게 된다. 따라서 프로그람안에서 기억기참조를 기호적으로 표시하고 다음에 콤파일 또는 아쎔블리할 때 그러한 기호참조들을 해결하도록 하는것이 좋다. 이것을 그림 7-15에서 보여 주고 있다. 명령이나 자료항목에 대한 매개 참조는 기호에 의하여 초기에 제시된다. 절대적재기에 의하여 모듈을 입력하기 위한 준비를 하면서 아쎔블러나 콤파일러는 주소들을 변환한다(이 실례에서는 적재하려는 모듈을 위치 1024로부터 시작한다.).

재배정가능한 적재

적재에 앞서 특정한 주소들에로 기억기참조들을 맺는것과 불리한 점은 결과적인 적재모듈을 주기억기의 한개의 령역에만 배치할수 있다는것이다. 그러나 많은 프로그람들이 주기억기를 공유할 때 특정한 모듈이 기억기의 어느 구역에 적재되겠는가를 앞질러서 결정하기는 힘들다. 그것을 판단하는것은 적재할 때 하는것이 더 좋다. 이때 적재모듈은 주기억기안의 임의의 곳에 있을수 있다.

이러한 요구를 만족시키기 위하여 아쎔블러나 콤파일러는 실제의 주기억주소(절대주소)가 아니라 프로그람의 시작점과 같이 알려진 어떤 점에 관계되는 주소를 만든다. 이런 수법을 그림 7-15 ㄷ 에서 설명하고 있다. 적재모듈의 시작점은 상대주소 0에 할당하고 모듈안의 다른 모든 기억기참조들은 모듈의 시작점에 상대적으로 표시한다.

모든 기억기참조들을 상대적인 형식으로 표시하면 적재기가 모듈을 원하는 위치에 배치하는 과제가 간단하게 된다. 모듈이 x위치로부터 시작하여 적재된다면 적재기는 모듈

을 기억기에 적재할 때 매 기억기참조에 단순히 x를 더하면 된다. 이 과제를 방조하기 위하여 적재모듈은 적재기에 주소참조가 어디에 있고 그것들이 어떻게 해석되는가(보통 프로그람의 원점에 대하여 상대적일수도 있다.)를 알려 주는 정보를 포함하여야 한다. 이러한 정보모임은 콤파일러나 아쎔블러가 준비하는데 이것을 보통 재배정사전이라고 한다.

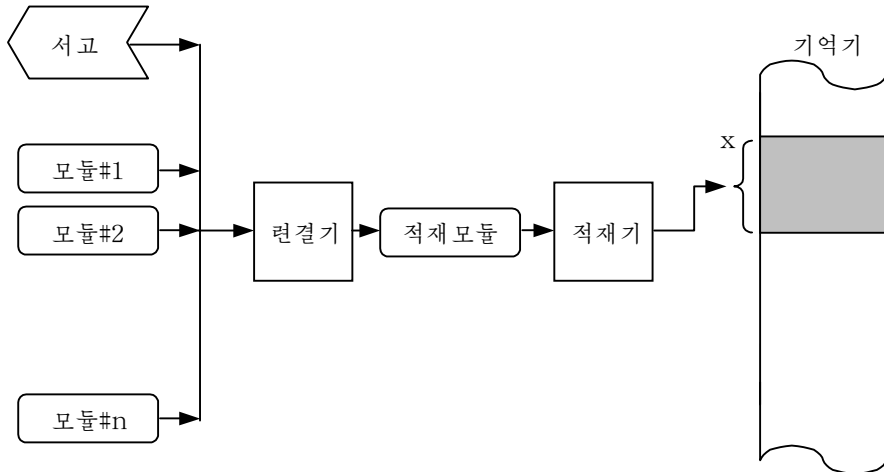



그림 7-14. 적재구성안

동적실행시적재

재배정가능한 적재기들은 일반적이고 절대적재기에 비하여 명백한 리득을 준다. 그러나 다중프로그람처리환경에서는 가상기억에 의거하지 않더라도 재배정가능한 적재방안이 적당하지 못하다. 처리기의 사용률을 최대한으로 높이기 위하여 주기억기의 안팎으로 프로쎄스형태를 교체할 필요가 있다. 주기억기의 사용률을 최대로 높이기 위하여 프로쎄스형태를 각이한 시간에 각이한 위치에 도로 교체할수도 있다. 이때 일단 적재된 프로그람은 디스크에 교체되여 나가고 다음에 다른 위치에 도로 교체하여 넣을수 있다. 이것은 초기적재시에 기억기참조들이 절대주소들과 엉켜 졌다면 불가능하다.

표 7-2. 주소맺기

ㄱ) 적재기

맺기시간	기능
프로그람작성시간	모든 실제물리주소들은 프로그람자체내에서 프로그람 작성자에 의하여 직접 명시된다.
콤파일러 또는 아쎔블러시간	프로그람은 기호주소참조들을 포함하며 이것들은 콤파일러나 아쎔블러에 의하여 실제적인 물리주소로 변환된다.
적재시간	콤파일러나 아쎔블러는 상대주소들을 발생시킨다. 적재기는 이것을 프로그람적재시에 절대주소로 변환한다.
실행시간	적재된 프로그람은 상대주소를 보유한다. 이것들은 처리기 하드웨어에 의하여 동적으로 절대주소들로 변환된다.

표계속

ㄴ) 련결기

련결시간	기 능
프로그람작성시간	외부프로그람이나 자료참조들은 허락되지 않는다. 프로그람 작성자는 참조되는 모든 부분프로그람들에 대한 원천코드를 프로그람에 배치하여야 한다.
콤파일러 또는 아셈블러 시간	아셈블러는 참조되는 매개 부분프로그람의 원천코드를 불러내여 어떤 단위로 아셈블리한다.
적재모듈창조	모든 목적모듈들은 상대주소들을 사용하여 아셈블리된다. 이 모듈들은 서로 련결되며 모든 참조들은 최종적재모듈의 원점에 대하여 재시동한다.
적재시간	외부참조들은 적재모듈이 주기억기에 적재될 때까지 해결되지 않는다. 그때 참조되는 동적련결모듈들은 적재모듈에 첨부되여 옹근제품이 주기억기 또는 가상기억기에 적재된다.
실행시간	외부참조들은 외부호출이 처리기에서 실행될 때까지 해결되지 않는다. 그때 프로세스는 새치기되며 요구되는 모듈이 호출중인 프로그람에 련결된다.

다른 방법은 실행할 때 실제적으로 필요하게 될 때까지 절대주소의 계산을 연기하는것이다. 이러한 목적으로 적재모듈은 상대적인 형식의 모든 기억기참조들을 가지고 주기억기에 적재된다(그림 7-15ㄷ). 명령이 실제로 실행될 때까지 절대주소는 계산되지 않는다. 이러한 기능이 성능을 떨어 뜨리지 않도록 하기 위하여 쏘프트웨어가 아니라 전용처리기하드웨어로 그것을 수행하여야 한다. 이 하드웨어에 대하여서는 제7장 제2절에서 서술하였다.

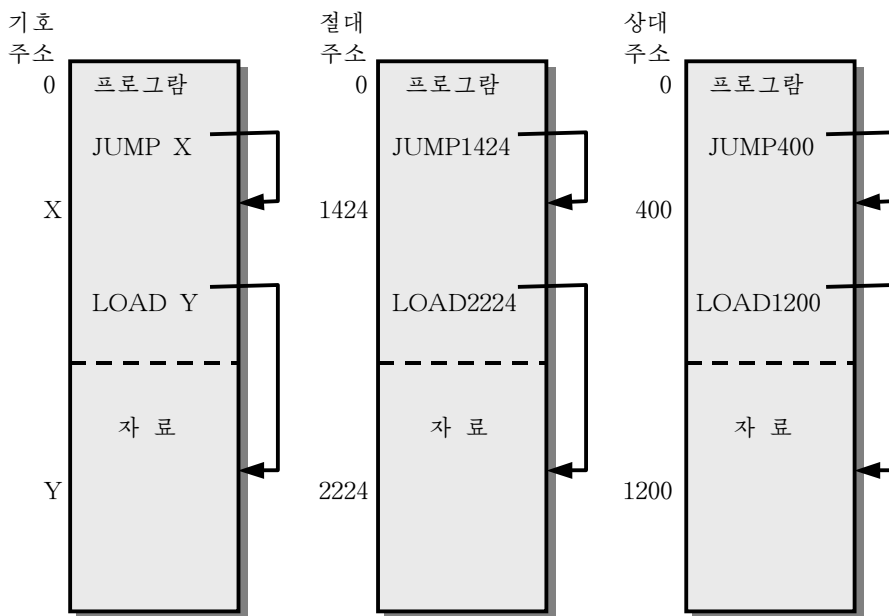



그림 7-15. 절대 및 재배정가능한 적재모듈
ㄱ-목적모듈, ㄴ-절대적재모듈, ㄷ-상대적재모듈

동적주소변환은 완전한 유연성을 가진다. 프로그람은 주기억기의 임의의 구역에 적재될수 있다. 그다음에는 프로그람실행을 새치기할수 있으며 프로그람도 주기억기밖으로 교체하여 내 보낼수 있고 후에 다른 위치에 도로 교체하여 넣을수 있다.

련결

련결기의 기능은 목적모듈들의 모임을 입력하여 프로그람과 자료모듈의 종합적인 모임으로 이루어 진 적재모듈을 생성하며 적재기에 넘겨 주는것이다. 매개 목적모듈에는 다른 모듈에 있는 위치에 대한 주소참조들이 있을수 있다. 이러한 매개 참조는 련결되지 않은 목적모듈에서 기호적으로만 표시될수 있다. 련결기는 모든 목적모듈들을 련속적으로 묶어서 한개의 옹근 적재모듈을 만든다. 모듈안의 매개 참조들은 기호주소로부터 총체적인 적재모듈안의 위치에 대한 참조로 변화시켜야 한다. 실례로 그림 7-16 ㄱ에서 모듈 A는 모듈 B의 수속요청을 포함한다. 모듈들이 적재모듈로 결합될 때 이 모듈에 대한 기호참조는 적재모듈안에서 B의 입구점위치에 대한 특정한 참조로 변화된다.

련결편집기

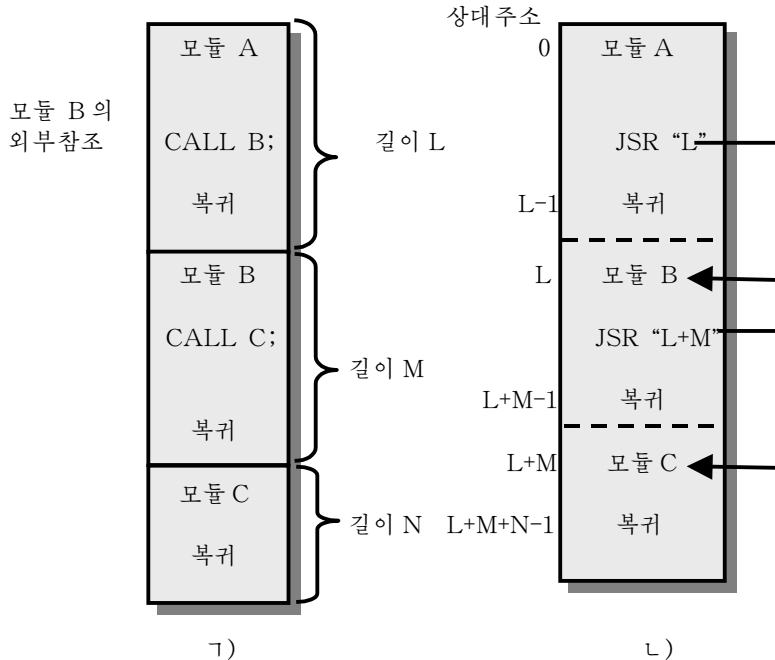
주소련결의 성질은 창조되는 적재모듈의 형태와 언제 련결이 진행되는가에 관계된다(그림 7-2 ㄴ). 일반적인 경우 재배정가능한 모듈이 요구될 때 련결은 보통 다음과 같은 형식으로 진행된다. 콤파일 또는 아쎔블리된 매개 목적모듈들은 목적모듈의 원점에 상대적인 참조를 가지고 창조된다. 이 모듈들은 모두 적재모듈의 원점에 상대적인 모든 참조를 가지고 하나의 재배정가능한 적재모듈에 다같이 넣어 진다. 이 모듈은 재배정가능한 적재나 동적실행시적재를 위한 입력으로 사용될수 있다. 재배정가능한 적재모듈을 만들어 내는 련결기를 흔히 련결편집기라고 한다. 그림 7-16에서는 련결편집기의 기능을 설명하고 있다.

동적련결기

적재할 때 일부 련결기능들을 지연시킬수 있다. 동적련결이라는 술어는 일부 외부모듈들의 련결을 적재모듈이 창조된 후에까지 연기하는것과 관련하여 사용한다. 이때 적재모듈은 다른 프로그람에 대한 해결되지 않은 참조들을 포함한다. 이 참조들은 적재시에나 실행시에 해결될수 있다.

적재시 동적련결에는 다음의 단계들이 있다. 우선 적재되여야 하는 적재모듈(응용프로그람모듈)을 기억기에 읽어 들인다. 다음에 외부모듈(목표모듈)에 대한 임의의 참조는 적재기가 목표모듈을 찾아 내고 그것을 적재하고 응용프로그람모듈의 시작점으로부터 기억기상의 상대주소에 대한 참조를 변경한다. 이 방법은 정적적재방법에 비하여 여러가지 우점을 가진다.

- 목표모듈의 변경판이나 갱신판을 만들기가 쉬워 지는데 그것은 조작체계의 유틸리티 또는 일부 다른 법용의 루틴일수 있다. 정적련결에서 지원모듈을 변화시키려면 응용프로그람모듈전체를 다시 련결하여야 하였다. 이것은 효률적이지 못할뿐아니라 어떤 경우에는 불가능할수도 있다. 실례를 들어 개인용콤퓨터환경에서 대부분의 상업용쏘프트웨어는 적재모듈형식에서 해방되고 원천과 오브젝트판은 해방되여 있지 않다.
- 동적련결파일에서 목표코드를 가지면 자동적인 코드공유를 할수 있다. 조작체계는 한개이상의 응용프로그람이 동일한 목표코드를 사용하는것을 인식할수 있다. 그것은 조작체계가 코드를 적재하고 련결하기때문이다. 매개 응용프로그람마다 목표코드를 한개씩 복사하여 적재하기보다 목표코드를 한개만 복사하여 적재하고 그것을 두개의 응용프로그람에 련결하면 그 정보를 쓸수 있다.

그림 7-16. 련결기능: ㄱ-목적모듈, ㄴ-적재모듈

- 독자적인 쏘프트웨어개발자들이 Linux와 같이 널리 사용되는 조작체계의 기능을 확장하기 쉽다. 개발자들은 여러가지 종류의 응용프로그람에 쓸모 있고 그것을 동적련결모듈로서 묶을수 있는 새로운 기능을 가지고 수준을 높일수 있다.

실행시동적련결에서는 일부 련결이 실행될 때까지 미루어 진다. 목표모듈에 대한 외부참조들은 적재된 프로그람에서 유지된다. 부재모듈에 대한 호출을 진행할 때 조작체계는 그 모듈을 찾아 적재하며 호출하는 모듈에 그것을 련결한다.

앞에서 동적인 적재가 전체 적재모듈을 움직이게 한다는것을 보았다. 그러나 모듈의 구조는 정적인것으로서 프로쎄스의 실행과정과 한 실행으로부터 다음 실행으로 넘어갈 때까지 변화되지 않는다. 그러나 일부 경우에 실행에 앞서 어느 목적모듈에 요구할것인가를 결정하는것이 불가능할수 있다. 이러한 정황은 비행기예약체계나 은행업무프로그람과 같은 거래처리응용프로그람들에서 전형적이다. 거래의 특징은 어떤 프로그람모듈들이 요구되는가를 지시하여 그 모듈들을 적당한 기회에 적재하고 주프로그람과 련결하는것이다. 이런 동적련결기를 사용하는것의 우점은 프로그람단위들이 참조되지 않는한 그것들을 위해 기억기를 배정할 필요가 없는것이다. 이런 능력은 토막화체계를 지원하는데 사용한다.

한가지 추가적인 개선방안이 있을수 있다. 응용프로그람은 모든 모듈들이나 호출될수 있는 입구점들의 이름들을 알 필요가 없다. 실례를 들어 지도작성프로그람은 매개가 서로다른 구동프로그람제품에 의해 구동되는 여러가지 종류의 작도기에서 쓰도록 작성될수 있다. 응용프로그람은 다른 프로쎄스로부터 또는 구성파일을 보고 체계에 현재 설치된 작도기의 이름을 얻을수 있다. 이것은 응용프로그람의 사용자가 응용프로그람이 작성될 때 존재하지 않았던 새로운 작도기를 설치할수 있게 해준다.

제 8 장. 가상기억기

제7장에서는 페지화와 토막화의 개념을 소개하고 그것들의 결합을 분석하였다. 이제부터 가상기억기에 대하여 설명한다. 가상기억기에 대한 분석은 기억기관리가 처리기하드웨어와 조작체계쏘프트웨어사이의 암시적이며 서로 얽혀 있는 호상관계로부터 복잡해진다. 우선 가상기억기의 하드웨어측면에 중심을 두고 페지화, 토막화 및 조합식페지화와 토막화에 대한 사용을 고찰한다. 다음으로 조작체계에서 가상기억기의 기억기능설계에 포함되는 문제들을 고찰한다.

제 1 절. 하드웨어 및 조종구조

단순한 페지화와 단순한 토막화를 비교하면서 한편으로 고정분할 및 동적분할을 비교하고 다른 한편으로는 기억기관리에서 제기되는 기본문제점들을 해결하기 위한 기초를 설명한다. 페지화와 토막화의 두가지 특성이 이 문제에서 열쇠로 된다. 즉

1. 프로쎄스안에서의 모든 기억기참조는 실행할 때 물리적주소에로 동적으로 변환되는 론리주소로 진행된다. 이것은 집행과정에 서로 다른 시간에 주기억기의 각이한 구역을 차지하도록 프로쎄스를 주기억기의 안팎으로 교체할수 있다는것을 의미한다.
2. 프로쎄스를 많은 쪼각(페지 또는 토막)들로 분할할수 있는데 이 쪼각들이 집행기간 전적으로 주기억기에 있어야 할 필요는 없다. 동적실행시 주소변환을 조합하고 페지 또는 토막표를 사용하면 이것을 실현할수 있다.

이제부터 이 문제점을 해명하기로 하자. 만일 우에서 지적한 두가지 특성이 존재한다면 프로쎄스의 모든 페지 또는 모든 토막들이 주기억기에 있을 필요는 없다. 만일 불러내야 할 다음명령을 포함하고 있는 쪼각(토막이나 페지) 및 주소지정하려는 다음자료위치를 포함하고 있는 쪼각이 주기억기에 있다면 적어도 일정한 시간동안은 집행을 계속할수 있다.

이것을 실현할수 있는 방법을 고찰하여 보자. 이제부터는 일반적인 용어로 말할수 있는데 페지화를 사용하는가 토막화를 사용하는가에 따라 쪼각이라는 용어를 페지나 토막이라는 말로 쓴다. 새로운 프로쎄스를 기억기에 가져 와야 할 시간이 되였다고 하자. 조작체계는 프로그람의 시작부를 담고 있는 쪼각을 가져 오기 위하여 한개 또는 몇개의 쪼각을 주기억기에 끌어 들이는것으로부터 시작한다. 임의의 시간에 실제적으로 주기억기에 있는 프로쎄스의 부분을 프로쎄스의 **상주모임**이라고 정의한다. 프로쎄스가 집행될 때 모든 기억기참조가 상주모임에 있는 기억위치에서 진행되는 동안은 조작이 순조롭게 진행된다. 토막 또는 페지표를 사용하여 처리기는 항상 이것이 어떻게 진행되는가를 결정한다. 만일 체계가 주기억기에 없는 론리적주소와 맞다들리면 새치기를 발생시켜 기억기접근부재를 알려 준다. 조작체계는 새치기된 프로쎄스를 페색상태에 놓고 조종을 차지한다. 프로쎄스의 집행을 후에 계속하기 위해 조작체계는 접근부재를 일으킨 론리주소를 담고 있는 프로쎄스의 쪼각을 주기억기에 가져다 넣는다. 이를 위해 조작체계는 디스크입출력읽기요청을 내보낸다. 입출력요청을 내보낸다음 조작체계는 디스크입출력이 수행

되는 동안에 또다른 프로쎄스를 배분하여 실행시킬수 있다. 일단 희망하는 쪼각이 주기억기에 들어왔으면 입출력새치기를 발행하고 조종을 조작체계에 돌려 주는데 이것은 영향받은 프로쎄스를 준비상태로 놓는다.

이 방식의 효과성에 대한 의문이 즉시에 제기될수 있는데 프로쎄스는 집행중에 있을수도 있고 프로쎄스의 필요한 모든 쪼각을 적재하는데서의 실패로 하여 새치기당할수도 있다. 이제부터 효과성이 있을수 있다는 확신을 가지고 이 문제에 대해 다르게 고찰해보자. 대신 새로운 전략에 대한 련관을 고찰하자. 두가지 련관이 있는바 두가지가 다 체계사용률을 개선시키지만 두번째 련관이 첫번째 련관보다 더 우월하다. 즉

1. **보다 많은 프로쎄스를 주기억기에 유지할수 있다.** 그것은 단지 특정한 프로쎄스의 일부 쪼각만을 적재시키기때문에 더 많은 프로쎄스를 위한 공간이 생기게 된다. 이것은 처리기의 사용률을 보다 효과적으로 높여 주는데 그것은 임의의 특정한 시각에 보다 많은 프로쎄스들중에서 적어도 하나가 준비상태에 있게 될 가능성이 더 커지기때문이다.
2. **프로쎄스가 주기억기의 전체보다 더 커질수 있다.** 프로그람작성에서 가장 기본적인 제약조건들중의 하나가 바로 이 문제이다. 지금까지 론의해 온 방안이 없다면 프로그람작성자는 사용가능한 기억기량을 민감하게 알고 있어야 한다. 작성되는 프로그람이 지내 길다면 프로그람작성자는 일정한 형태의 겹침방법을 써서 개별적으로 적재시킬수 있는 쪼각들로 프로그람을 구조화해야 한다. 페지화 또는 토막화에 기초한 가상기억기를 사용하면 그 일감은 조작체계와 하드웨어가 담당한다. 프로그람작성자가 관여하는 한 그 누구든지 디스크기억기와 같은 크기의 큰 기억기를 취급하게 된다. 조작체계는 요구에 따라 프로쎄스의 쪼각들을 자동적으로 주기억기에 적재한다.

프로쎄스가 주기억기에서만 집행되기때문에 그 기억기를 **실제기억기**라고 한다. 한편 프로그람작성자나 사용자는 디스크상에 배정되여 있는 잠재적으로 훨씬 더 큰 기억기를 생각하게 된다. 이것을 **가상기억기**라고 한다. 가상기억기는 매우 효과적으로 다중프로그람처리를 할수 있게 하며 사용자가 주기억기에 대해 불필요한 구속을 받지 않도록 해준다. 표 8-1 에서는 가상기억기를 사용하는 경우와 하지 않는 경우 페지화와 토막화의 특성들을 요약하여 설명해 주고 있다.

국소성과 가상기억기

가상기억기의 편리성은 매력적이지만 그 방안이 과연 실제적인가? 한때 이 점에 대해 많은 론의가 있었으나 많은 조작체계들에서의 경험은 가상기억기가 정말 동작한다는 것을 의심할바 없이 증명하였다. 따라서 그것은 대부분의 조작체계들에서 필수적인 구성요소로 되여 왔다.

가상기억기에서 기본문제가 무엇이며 왜 그렇게 많이 론의되였는가를 리해하기 위해 가상기억기와 관련한 조작체계의 과제를 다시 검토해 보자. 긴 프로그람과 또 많은 자료배렬로 이루어 지는 규모가 큰 프로쎄스를 고찰하자. 어떤 짧은 시간주기동안에는 집행이 프로그람의 작은 구역(실례로 보조루틴)에만 국한될수 있으며 혹시 한개 또는 두개의 자료배렬에만 접근할수 있다. 만일 그렇게 된다면 프로그람이 중단되고 교체되여 나가기전에 몇개의 쪼각들만 사용하면 되므로 이때 프로쎄스를 위하여 수십개의 쪼각을 적재시키는것은 명백히 랑비로 된다. 바로 몇개의 쪼각들만 적재시킴으로써 기억기를 더 잘 사용할수 있다.

표 8-1. 페지화와 토막화의 특성

단순페지화	가상기억기페지화	단순토막화	가상기억기토막화
주기억기가 프레임이라고 하는 작은 고정크기의 덩어리로 분할된다.	주기억기가 프레임이라고 하는 작은 고정크기의 덩어리로 분할된다.	주기억기는 분할되지 않는다.	주기억기가 분할되지 않는다.
프로그람이 콤파일러나 기억기관리체계에 의해 페지로 분할된다.	프로그람이 콤파일러나 기억기관리체계에 의해 페지로 분할된다.	프로그람토막들이 프로그람작성자에 의해 콤파일러에 명시된다.(즉 프로그람작성자가 결정한다.)	프로그람토막들이 프로그람작성자에 의해 콤파일러에 명시된다.(즉 프로그람작성자가 결정한다.)
프레임이내부쪼각화된다.	프레임이 내부쪼각화된다.	내부쪼각이 없다.	내부쪼각이 없다.
외부쪼각이 없다.	외부쪼각이 없다.	외부쪼각이 있다.	외부쪼각이 있다.
조작체계는 매개 페지가 어느 프레임을 차지하는가를 보여 주기 위해 매개프로쎄스에 대한 페지표를 유지하여 야 한다.	조작체계는 매개 페지가 어느 프레임을 차지하는가를 보여 주기 위해 매개 프로쎄스에 대한 페지표를 유지하여야 한다.	조작체계가 매개 토막의 적재주소와 길이를 보여 주기 위하여 매개 프로쎄스의 토막표를 유지하여야 한다.	조작체계가 매개 토막의 적재주소와 길이를 보여 주기 위하여 매개 프로쎄스의 토막표를 유지하여야 한다.
조작체계가 자유프레임목록을 유지하여야 한다.	조작체계가 자유프레임목록을 유지하여야 한다.	조작체계가 주기억기에서 자유구멍목록을 유지해야 한다.	조작체계가 주기억기에서 자유구멍목록을 유지해야 한다.
처리기가 페지번호,편위를 사용하여 절대주소를 계산한다.	처리기는 절대주소를 계산하기 위하여 페지번호,편위를 사용한다.	처리기가 절대주소를 계산하기 위하여 토막번호,편위를 사용한다.	처리기가 절대주소를 계산하기 위하여 토막번호,편위를 사용한다.
겹침을 사용하지 않는한 프로쎄스의 모든 페지는 프로쎄스가 실행할 주기억기에 있어야 한다.	프로쎄스의 모든 페지가 프로쎄스가 실행되는 주기억기프레임들에 있을 필요는 없다. 필요에 따라 페지들을 읽어 들일수 있다.	겹침을 사용하지 않는한 프로쎄스의 모든 토막들은 프로쎄스가 실행할 주기억기에 있어야 한다.	프로쎄스의 모든 토막이 프로쎄스가 실행되는 주기억기프레임들에 있을 필요는 없다. 필요에 따라 토막들을 읽어 들일수 있다.
	주기억기에로의 페지읽기는 디스크밖으로의 페지쓰기를 요구할수 있다.		주기억기에로의 토막읽기는 디스크밖으로의 한개 또는 그이상의 토막쓰기를 요구할수 있다.

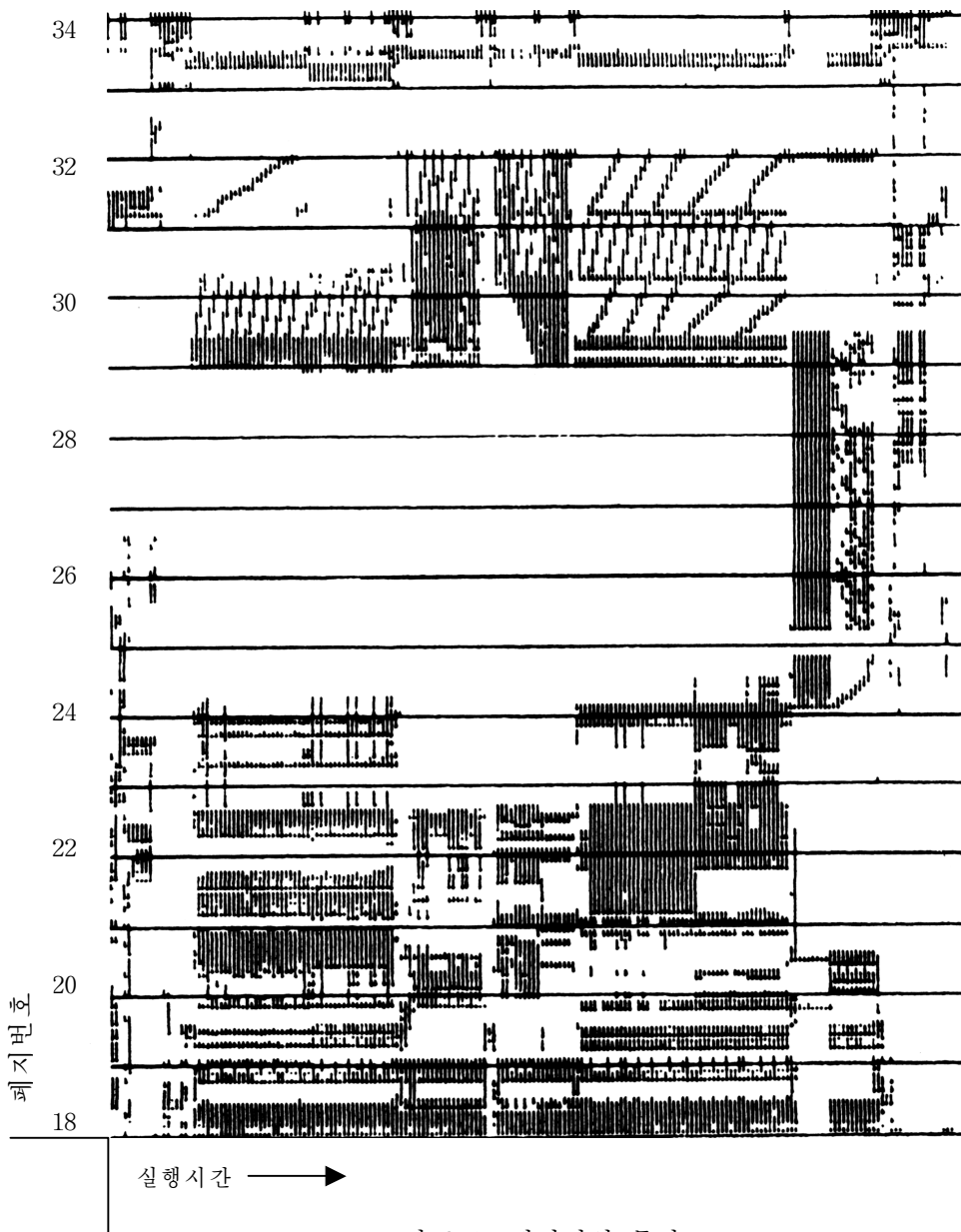



그림 8-1. 페지화의 동작

만일 프로그람이 주기억기에 없는 쪼각의 명령에로 갈래를 일으키든가 자료항목을 참조한다면 부재오유가 발생한다. 그러면 조작체계가 요구하는 쪼각을 끌어 들이도록 한다.

그러므로 임의의 시각에 주어 진 프로쎄스의 몇개의 쪼각만이 주기억기에 있게 되며 따라서 보다 많은 프로쎄스를 기억기에 유지할수 있다. 더우기 사용되지 않는 쪼각들이 기억기에로 교체되여 들어 오거나 나가지 않으므로 그만큼 시간이 절약된다. 그러나 조작체계는 이 방안을 관리하는 방법을 재치있게 해내야 한다. 안정상태에서 실제적으로 모든 주기억기는 프로쎄스쪼각들이 차지하며 결국 처리기와 조작체계는 가능한한 많은

프로쎄스들에 직접 접근한다. 그러므로 조작체계가 한 쪼각을 가져다 넣을 때 다른 한 쪼각을 내보내야 한다. 만일 방금 막 사용하려고 하는 찰나에 어떤 쪼각을 내보내게 된다면 거의나 즉시에 그 쪼각을 다시 가서 가져 와야 할것이다. 이것이 지내 많아 지면 **파도교체**라는 조건이 생긴다. 처리기는 대부분의 시간을 명령을 집행하는데가 아니라 쪼각들을 교체하는데 소비한다. 파도교체현상을 피하는것이 1970년대의 주되는 연구분야로 되였고 여러가지의 복잡하나 효과적인 알고리듬이 나왔다. 본질상 조작체계로 하여금 최근 경력에 기초하여 어느 프로쎄스가 가까운 장래에 최소한 사용됨직한가 하는것을 추측하게 하는데 있다.

이것은 **국소성의 원리**에 기초한것으로서 제1장에서 소개되였다(특히 부록 1-ㄱ를 볼것). 요약해서 말한다면 국소성의 원리는 프로쎄스의 프로그람과 자료의 참조가 클라스터를 짓는 경향성을 가진다는것을 설명하고 있다. 이로부터 짧은 시간동안에는 프로쎄스의 몇개의 쪼각만이 요구될것이라는 가정이 유효하다. 또한 프로쎄스의 어느 쪼각이 가까운 장래에 필요되겠는가 하는것을 지능적으로 추측할수 있으며 이것은 파도교체현상을 피하게 해준다.

국소성의 원리를 확증하는 한가지 방법은 가상기억기환경에서 프로쎄스의 성능을 보는것이다. 그림 8-1은 국소성의 원리를 훌륭히 설명해 주는 비교적 유명한 선도이다[HATF72]. 프로쎄스의 수명기간에 참조는 페지의 부분모임에 국한된다.

그러므로 국소성의 원리가 작용하는 가상기억기방안을 제기한다. 가상기억기가 실제적이며 효과적인것으로 되자면 두가지 구성요소가 필요하다. 우선 페지화와 토막화를 사용할수 있게 하는 하드웨어가 있어야 한다. 둘째로, 조작체계는 2 차기억기와 주기억기사이에서 페지와 토막의 이동을 관리하는 쏘프트웨어를 가지고 있어야 한다. 이 절에서는 하드웨어측면을 고찰하고 필요한 조종구조를 보게 되는데 이것은 조작체계가 창조하고 유지하지만 기억기관리하드웨어가 사용한다. 조작체계의 문제점들에 대한 고찰은 다음 절에서 한다.

페지화

가상기억기라는 용어는 비록 토막화에 기초한 가상기억기에서 쓰이고 있으나 보통은 페지화를 사용하는 체계와 관련되여 있다. 페지화를 사용하여 가상기억기를 실현하는것은 처음으로 Atlas의 콤퓨터 [KILB62]에서 제기되였고 곧 상업적사용에 널리 보급되였다.

단순페지화에서는 매개 프로쎄스가 자체의 페지표를 가지며 그것의 모든 페지가 주기억기에 적재될 때 프로쎄스의 페지표가 창조되여 주기억기에 적재된다. 매개 페지표입구점은 주기억기의 대응하는 페지프레임번호를 포함하고 있다. 동일한 장치, 페지표는 페지화에 기초하고 있는 가상기억기방안을 고찰할 때 필요하다. 또 단일한 페지표를 매개 프로쎄스와 련관시키는것이 일반적이다. 그러나 이 경우에 페지표입구점들은 보다 복잡해 진다(그림 8-2 ㄱ). 프로쎄스의 일부 페지만이 주기억기에 있을수 있으므로 매개 페지표입구점에 한개 비트가 있어서 대응하는 페지가 주기억기에 있는가(P) 없는가를 지적해 주어야 한다. 만일 그 비트가 해당 페지가 기억기에 있다고 지적하면 입구점은 또한 그 페지의 프레임번호를 포함한다.

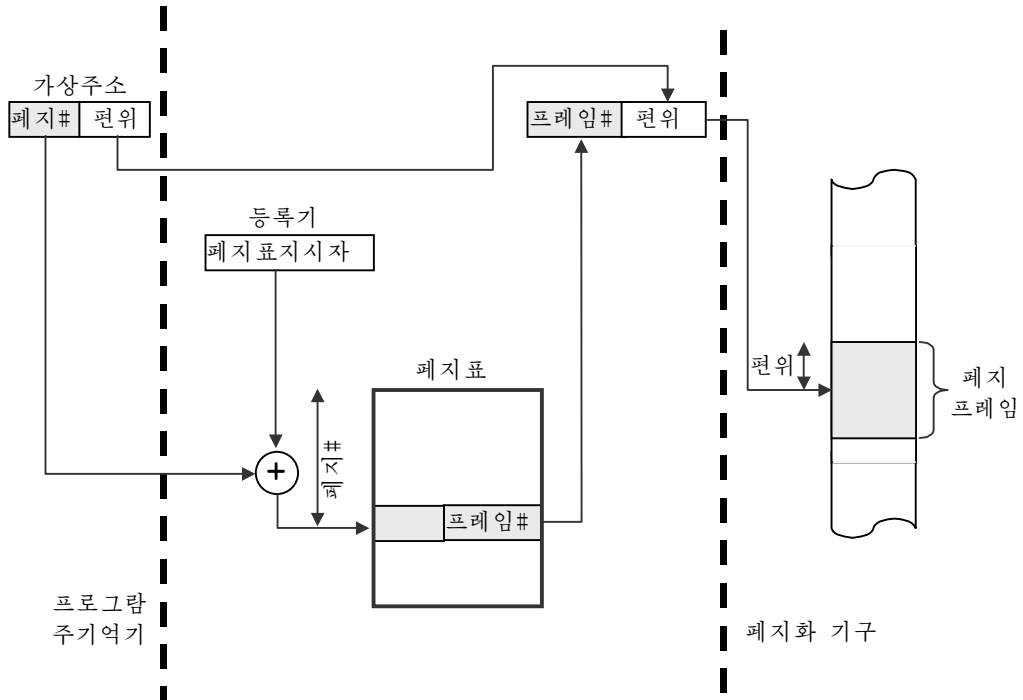
페지표입구점에 필요한 또 하나의 조종비트는 변경(M)비트로서 대응하는 페지의 내용이 그 페지가 주기억기에 마지막으로 적재된 때로부터 변경되였는가 하는것을 지적해준다. 현재 차지하고 있는 프레임에서 페지를 치환해야 할 시간이 되였을 때 만일 아무런 변화도 없었으면 그 페지를 외부쓰기할 필요가 없다. 다른 조종비트들도 존재한다. 실례로 페지준위에서 보호나 공유를 관리한다면 그 목적을 위한 비트들이 요구된다.

그림 8-2. 대표적인 기억기관리의 양식
ㄱ- 유일페지화, ㄴ-유일토막화, ㄷ-조합식토막화 및 페지화

페지표구조

기억기에서 단어를 읽는 기본기구는 페지표를 사용하여 페지번호와 편위로 구성되여 있는 가상주소 즉 론리적주소를 프레임번호와 편위로 구성되여 있는 물리적주소로 변환하는 과정을 동반한다. 프로쎄스의 크기에 따라 페지표의 길이가 변하기때문에 그것을 등록기에 유지할수는 없다. 대신 그것에 접근할수 있도록 기억기에 보관해야 한다. 그림 8-3에서는 하드웨어실현을 보여 주고 있다. 특정한 프로쎄스가 실행하고 있을 때 등록기는 프로쎄스에서의 페지표시작주소를 보존한다. 가상주소의 페지번호는 그 표를 지시하는데 사용되며 대응하는 프레임번호를 찾는데 쓰인다.

대부분의 체계들에는 프로쎄스당 한개의 페지표가 있다. 그러나 매개 프로쎄스는 많은 가상기억기를 차지할수 있다. 실례로 VAX구성방식에서 매개 프로쎄스는 2^{31}=2Gbyte까지의 가상기억기를 가질수 있다. 2^{9}=512byte의 페지를 사용한다는것은 프로쎄스마다 2^{22} 만한 페지표입구점이 필요하다는것을 의미한다. 명백히 페지표들에 들어 가는 기억기량은 받아 들일수 없을 정도로 많아 질수 있다. 이 문제를 극복하기 위하여 대부분의 가상기억기방안들에서는 페지표를 실제기억기가 아니라 가상기억기에 기억시킨다. 이것은 페지표들이 다른 페지가 영향 받는것처럼 페지화의 영향을 받는다는것을 의미한다.

그림 8-3. 페지화체계에서 주소변환

프로쎄스가 실행중에 있을 때 적어도 그 페지표의 일부라도 주기억기에 있어서 현재 집행중인 페지의 페지표입구점을 포함하고 있어야 한다. 일부 처리기들은 2준위방안을 사용하여 큰 페지표를 작성할수 있다. 이 방안에는 페지등록부가 있는데 거기서 매개 입구점은 페지표를 가리킨다. 그러므로 페지등록부의 길이가 X이고 페지표의 최대길이가 Y라면 프로쎄스는 $X \times Y$까지의 페지로 구성할수 있다. 대체로 페지표의 최대길이는 한개 페지와 같아 질 정도로 제한된다. 실례로 펜티움처리기가 바로 이 방법을 사용하고 있다.

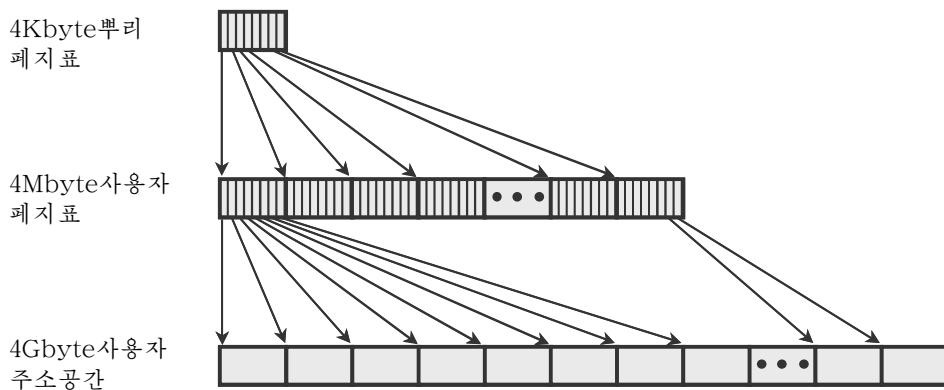



그림 8-4. 2준위계층형페지표[JACO98a]

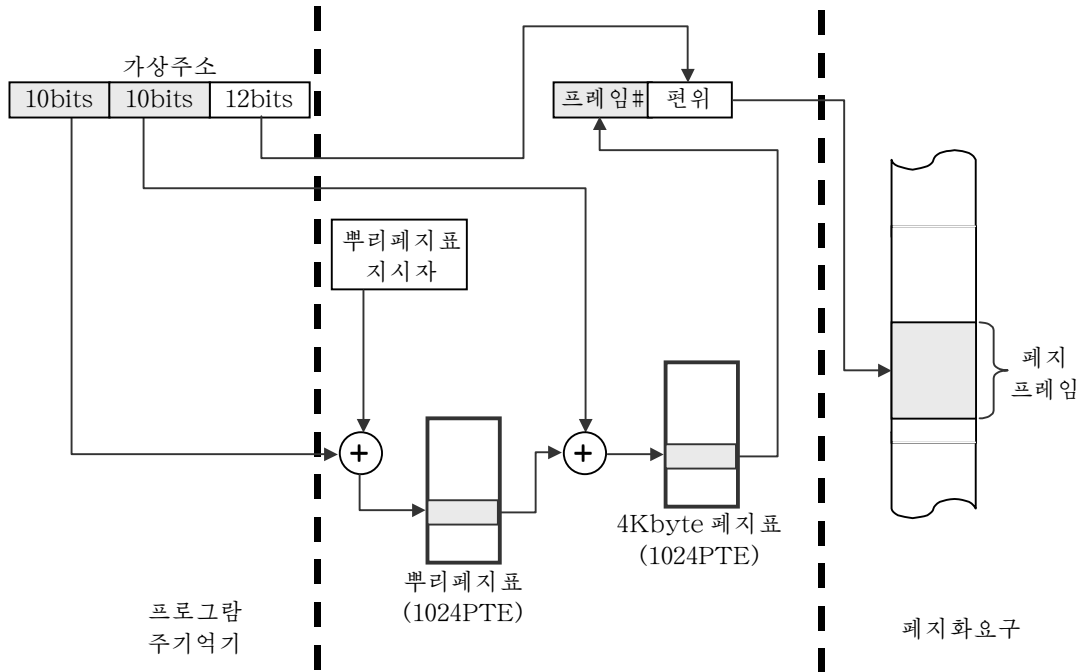



그림 8-5. 2 준위페지화체계에서의 주소변환

그림 8-4에서는 32bit 주소를 사용하는데서 대표적인 2준위방안에 대한 실례를 보여주고 있다. 만일 바이트준위주소방식과 4kbyte(2^{12})의 페지들을 가상한다면 4Gbyte(2^{32})의 가상주소공간이 2^{20} 개의 페지들로 구성된다. 만일 매개 페지가 4byte 의 페지표입구점(PTE)에 의해 사영된다면 4Mbyte(2^{22}byte)를 요구하는 2^{20}개의 PTE 로 구성되는 사용자페지표를 창조할수 있다. 2^{10}페지로 구성되는 큰 사용자페지표를 가상기억기에 유지할수 있으며 4kbyte(2^{12})의 주기억기를 차지하는 2^{10}PTE 를 가진 뿌리페지표를 써서 사영할수 있다. 그림 8-5에서는 이 방안에서의 주소변환에 포함된 단계들을 보여 준다. 뿌리페지는 주기억기에 항상 남아 있는다. 가상주소의 첫 10bit 는 뿌리페지에 들어 가 사용자페지표의 어떤 페지용 PTE 를 찾기 위한 첨수로 쓰인다. 만일 페지가 주기억기에 없다면 페지부재가 발생한다. 페지가 주기억기에 있으면 가상주소의 다음 10bit 는 사용자의 PTE 페지에로 첨수주소화하여 가상주소가 참조하는 페지용 PTE 를 찾는다.

1 또는 2 준위페지표의 사용을 대신할수 있는 방법은 **반전페지표**구조를 사용하는것이다(그림 8-6). 이 방법은 Power PC 및 IBM 의 AS/400 에서 쓰이고 있다. RT-PC 상에서 Mach 조작체계가 또한 이 수법을 사용하고 있다.

이 방법에서 가상주소의 페지번호부분은 단순하쉬기능[1]을 사용하는 하쉬표에로 사영된다. 하쉬표는 반전페지표에 대한 지시기를 포함하고 있는데 이것은 페지표입구점을 가지고 있다. 이 구조를 보면 가상페지당 한개라기보다 매개 실제기억기페지용으로 하쉬표와 반전페지표안에 한개의 입구점이 있다. 그러므로 지원되는 프로쎄스나 가상페지수에 관계 없이 페지용으로 실제기억기의 고정된 부분이 필요하다. 하나이상의 가상주소가 같

[1] 하쉬법에 대해서는 부록 8-ㄱ를 보시오.

은 하쉬표입구점에 사영할수 있기때문에 자리넘침관리에 사슬수법을 사용한다. 하쉬수법은 1과 2입구점사이에서 대체로 짧은 사슬을 요구한다.

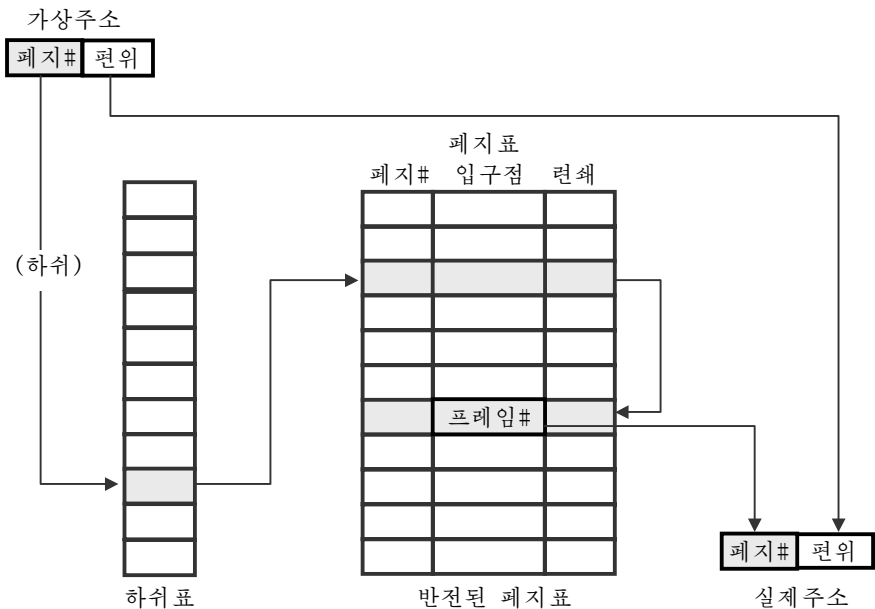



그림 8-6. 반전페지표구조

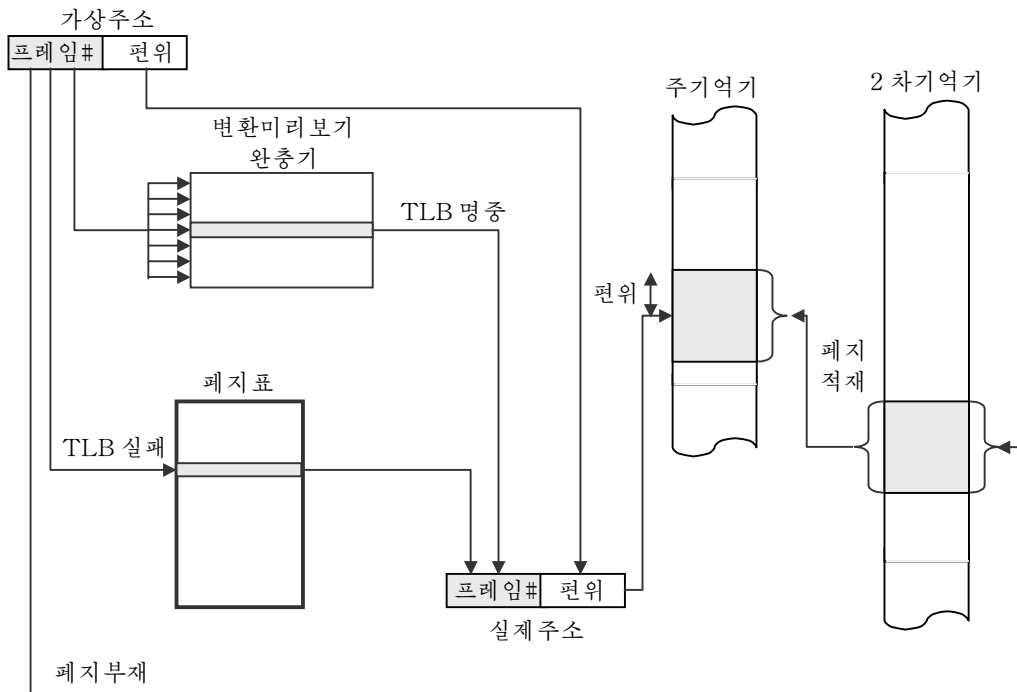



그림 8-7. 변환미리보기완충기의 사용

변환미리보기완충기

원리적으로 매개 가상기억기참조는 두가지 물리적기억기접근을 발생시킬수 있다. 즉 하나는 해당한 페지표입구점을 불러 내는것이고 다른 하나는 요구하는 자료를 불러 내는 것이다. 그러므로 직선적인 가상기억기방안은 기억기접근시간을 두배로 취하는 효과를 가지게 될것이다. 이 문제를 극복하기 위해 대부분의 가상기억기방안들에서는 일반적으로 **변환미리보기완충기**(TLB)라고 부르는 페지표입구점용으로 특수한 고속캐쉬를 사용한다. 이 캐쉬는 기억기캐쉬와 같은 방법으로 동작하며(제 1 장을 보시오.) 가장 최근에 사용한 페지표입구점을 가지고 있다. 페지화하드웨어의 구성을 그림 8-7 에서 설명하고 있다. 가상주소가 주어 지면 처리기는 우선 TLB를 조사한다. 요구되는 페지표입구점이 존재한다면(《TLB명중》) 프레임번호가 회복되여 실제주소가 형성된다. 요구되는 페지표입구점이 발견되지 않으면(《TLB실패》) 처리기는 페지번호를 사용하여 프로쎄스페지표를 첨수주소화하고 대응하는 페지표입구점을 조사한다. 만일 《존재비트》가 설정되여 있으면 페지가 주기억기에 있으며 처리기가 페지표입구점으로부터 프레임번호를 회복하여 실제주소를 형성할수 있다. 처리기는 새로운 페지표입구점을 포함하기 위하여 또한 TLB를 갱신한다.

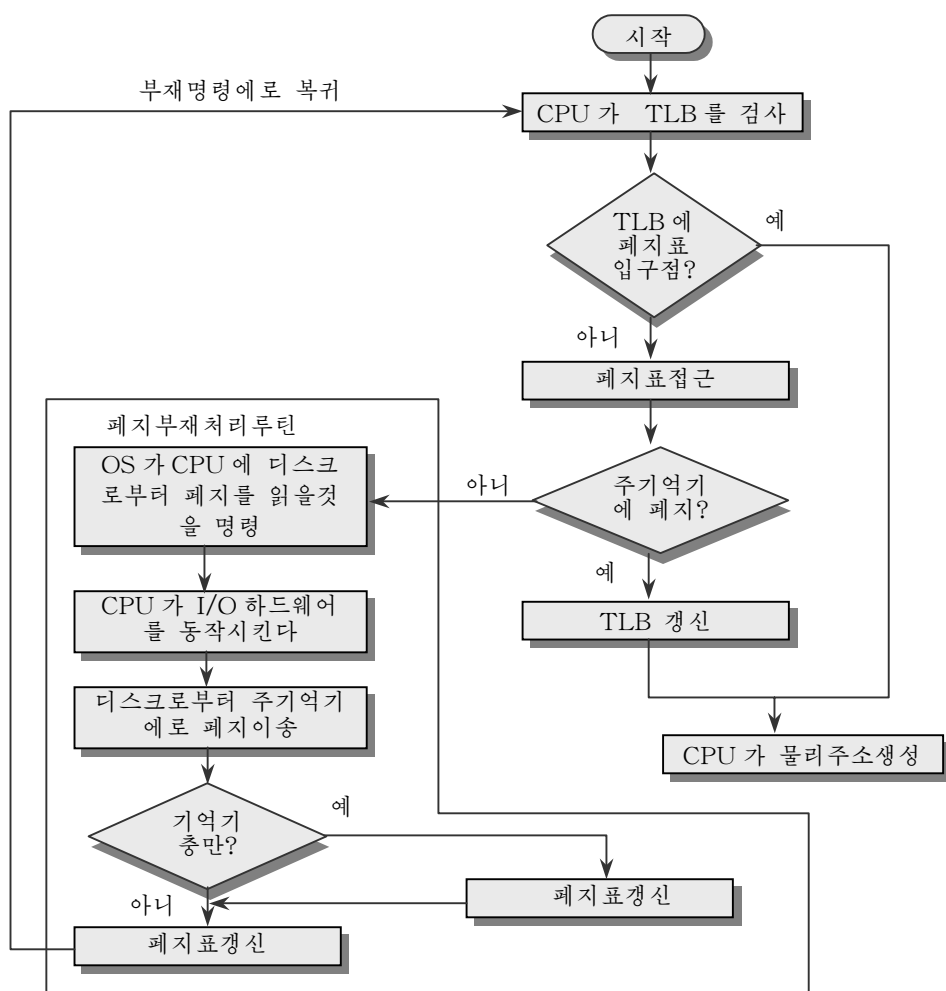



그림 8-8. 페지화 및 변환미리보기완충기의 조작[FURH87]

끝으로 존재비트가 설정되여 있지 않으면 요구되는 페지는 주기억기에 없으며 **페지부재**라고 부르는 기억기접근오유가 생긴다. 이 시점에서 하드웨어령역을 리탈하여 조작체계를 기동하는데 조작체계는 요구되는 페지를 적재하고 페지표를 갱신한다.

그림 8-8 은 TLB 의 사용을 보여 주는 흐름도이다. 흐름도는 요구하는 페지가 주기억기에 없으면 페지표부재새치기를 발생시켜 페지부재처리루틴을 기동하도록 한다는것을 보여 준다. 흐름도를 간단히 하기 위해 디스크입출력이 진행되는 동안에 조작체계가 또 다른 프로쎄스를 배분할수 있다는것은 보여 주지 않았다. 국소성의 원리로부터 대부분의 가상기억기참조는 최근에 사용한 페지에서 진행된다. 따라서 대부분의 참조는 캐쉬에 있는 페지표입구점들을 포함한다. VAX 의 TLB 들에 대한 연구결과는 이 방안이 성능을 현저히 개선할수 있다는것을 보여 주었다[CLAR85, SATY81].

TLB의 실제적구성에 대한 많은 추가적인 세부들이 있다. TLB는 총페지표안에서 다만 일부 입구점들을 포함하고 있으므로 페지번호에 기초하고 있는 TLB에로 단순하게 첨수주소화할수 없다. 대신 TLB안에 있는 매개 입구점은 페지번호는 물론 완전한 페지표입구점을 포함하고 있어야 한다. 처리기는 동시에 많은 TLB입구점들에 질문하여 페지번호와 맞는것이 있는가를 판정할수 있는 하드웨어를 장비하고 있다. 이 수법을 **련상사영**이라고 하며 그림 8-9 에서와 같이 페지표의 찾기에 사용되는 직접사영 또는 첨수주소화사영과 대조된다. TLB의 설계에서는 또한 TLB의 입구점들을 구성하는 방법과 새로운 입구점이 들어 오게 될 때 어느 입구점이 치환되는가를 고려하여야 한다. 이 문

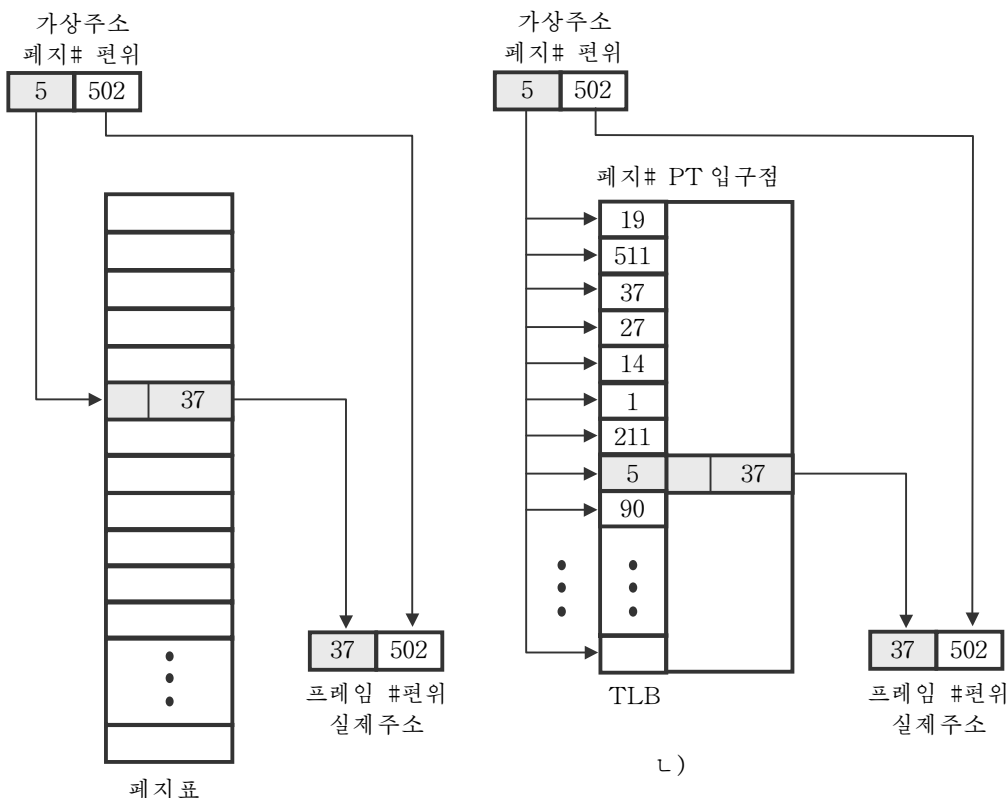



그림 8-9. 페지표입구점에서의 직접 대 련상미리보기
ㄱ-직접사영법, ㄴ-련상사영법

제들은 임의의 하드웨어캐쉬설계에서 고찰되여야 한다. 이 문제를 여기서 더 론하지 않는데 세부적인 캐쉬설계에 대해서는 다른 문헌에서 볼수 있다(실례로 [STAL00]).

끝으로 가상기억기기구는 캐쉬체계와 대화를 하여야 한다(TLB 캐쉬뿐아니라 주기억기캐쉬와도). 이것을 그림 8-10 에서 설명하고 있다. 가상주소는 일반적으로 페지번호, 편위형태로 되여 있다. 우선 기억기체계는 TLB 와 대화하여 일치하는 페지표입구점이 존재하는가를 본다. 만일 있다면 실제(물리적)주소가 프레임번호와 편위의 조합으로 생성된다. 그렇지 않다면 페지표로부터 입구점에 접근한다. 일단 실제주소가 발생되면 이것은 태그 [2] 및 나머지의 형식으로 되여 있는데 캐쉬와 대화하여 단어를 포함하는 블로크가 존재하는가를 본다. 그렇다면 그것이 CPU 에 반환된다. 그렇지 않다면 단어가 주기억기로부터 회복된다.

독자들은 단일기억기참조에 포함된 CPU 하드웨어의 복잡성을 알수 있을것이다. 가상주소는 실제주소로 변환된다. 이것은 페지표입구에 대한 참조를 포함하는데 TLB, 주기억기, 또는 디스크에 있을수 있다. 참조된 단어는 캐쉬, 주기억기 또는 디스크상에 있을수 있다. 후자의 경우에 단어를 포함하고 있는 페지는 주기억기에 적재되여야 하며 블로크는 캐쉬에 적재되여야 한다. 또한 페지에서의 페지표입구점이 갱신되여야 한다.

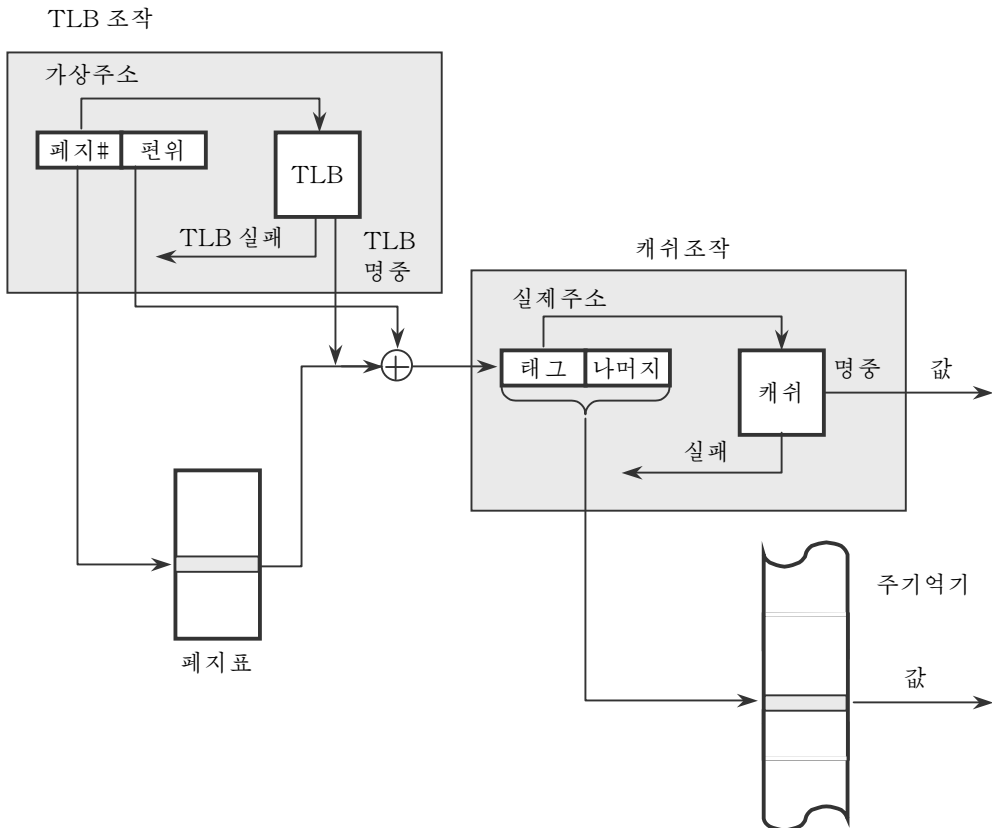



그림 8-10. 변환미리보기완충기와 캐쉬조작

---

2. 그림 1-17 을 보시오. 대표적으로 태그는 바로 실제주소의 제일 왼쪽의 몇개 비트이다. 또한 캐쉬에 대한 더 구체적인 설명에 대해서는 [STAL00]을 보시오.

페지크기

하드웨어설계에서 중요하게 결정해야 할 문제는 사용되는 페지의 크기이다. 몇가지 고려해야 할 인자들이 있다. 하나가 내부쪼각화이다. 명백히 페지크기가 작을수록 내부쪼각화의 량도 작아 진다. 주기억기의 사용을 최적화하자면 내부쪼각화를 줄이는것이 좋다. 다른 한편 페지가 작을수록 프로쎄스당 요구되는 페지의 수는 더 커진다. 프로쎄스당 페지가 더 많다는것은 페지표가 더 커진다는것을 의미한다. 무거운 다중프로그람처리환경에서 집행되는 규모가 큰 프로그람들에서 이것은 능동프로쎄스들의 페지표의 일부분이 주기억기에가 아니라 가상기억기에 있어야 한다는것을 의미한다. 그러므로 기억기에 대한 단일참조에서 2 중페지부재가 있을수 있다. 즉 첫째로, 요구되는 페지표의 일부를 가져다 넣는데서 그리고 둘째로, 프로쎄스의 페지를 가져다 놓는데서 있을수 있다. 또다른 인자는 대부분의 회전하는 2 차기억장치들의 물리적특성이 보다 효률적인 자료블로크이송을 위하여 더 큰 페지크기를 요구한다는것이다.

이 문제를 복잡하게 만드는것은 페지부재가 일어 나는 비률에 대한 페지크기의 영향이다. 이 성질을 일반적인 용어들로 그림 8-11 ㄱ에서 보여 주고 있는데 국소성의 원리에 기초하고 있다. 페지크기가 매우 작다면 보통 상대적으로 많은 페지들을 프로쎄스용으로 주기억기에서 사용할수 있을것이다. 시간이 지나면 기억기에 있는 페지들은 모두 최근에 참조한 근방에 있는 프로쎄스의 부분들을 포함할것이다. 그러므로 페지부재률은 낮아 진다. 페지의 크기가 증가됨에 따라 매개 개별적인 페지는 임의의 특정한 최소참조와 더욱더 멀리 떨어 진 위치들을 포함한다. 그러므로 국소성원리의 효과는 약해 지고 페지부재률은 올라 가기 시작한다. 그러나 결과적으로 페지부재률은 페지의 크기가 전체 프로쎄스의 크기에 다가감에 따라 떨어 지기 시작한다(선도에서 점 *P*). 단일한 페지가 전체 프로쎄스를 포괄하면 아무런 페지부재도 없을것이다.

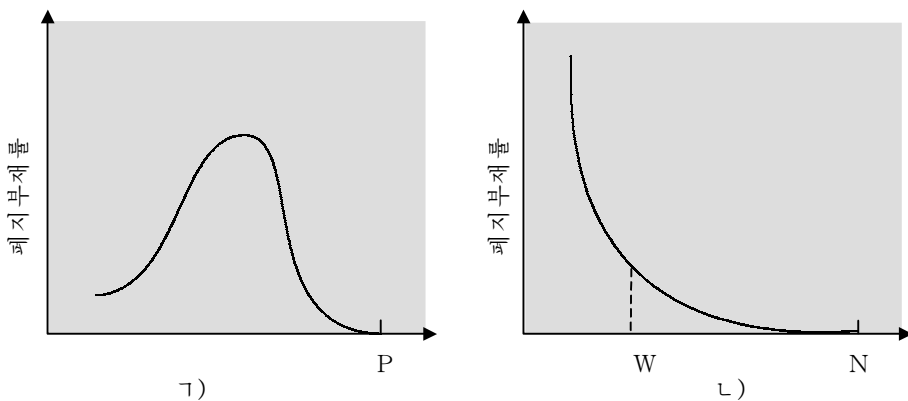
더우기 복잡한것은 페지부재률이 프로쎄스에 배정된 프레임의 수에 따라서도 결정된다는것이다. 그림 8-11 ㄴ가 그것을 보여 주고 있는데 고정된 페지크기에 대하여 부재률

P = 옹근프로쎄스의 크기
W = 작업모임의 크기
N= 프로쎄스내의 총 페지번호

그림 8-11. 프로그람의 대표적인 페지화의 성질
ㄱ-페지크기, ㄴ-배정된 페지프레임수

은 주기억기에 유지되여 있는 페지의 수가 커짐에 따라 떨어 진다.[3] 그러므로 쏘프트웨어방책(매개 프로쎄스에 배정되는 기억기량)은 하드웨어설계의 결정(페지크기)에 영향을 준다.

표 8-2에서는 일부 기계들에서 사용된 페지크기들을 지적하고 있다.

끝으로 페지크기의 설계문제는 물리적인 주기억기의 크기와 프로그람의 크기에 관련된다. 주기억기가 점점 커지는것과 동시에 응용프로그람들이 사용하는 주소공간도 늘어난다. 그런 경향성은 개인용콤퓨터와 작업기들에서 가장 명백한데 여기서 응용프로그람들은 아주 복잡해 지고 있다. 더우기 규모가 큰 프로그람들에서 사용된 당시의 프로그람작성수법들은 프로쎄스안에서 참조의 국소성을 감소시키는 경향성을 가지고 있다 [HUCK93]. 실례를 들면

- 객체지향수법들은 상대적으로 짧은 시간동안에 상대적으로 많은 수의 객체들에 대해 분산된 참조를 가진 많은 작은 프로그람과 자료모듈을 사용한다.
- 다중스레드식응용프로그람들은 명령흐름에서와 분산된 기억기참조에서 급격한 변화를 일으킬수 있다.

표 8-2. 페지크기에 대한 실례

콤퓨터	페지크기
Atlas	512개의 48bit 단어
Honeywell-Multics	1024개의 36bit 단어
IBM 370/XA 와 370/ESA	4kbyte
VAX 계렬	512byte
IBM AS/400	512byte
DEC Alpha	8kbyte
MIPS	4kbyte - 16Mbyte
Ultra SPARC	8kbyte - 4Mbyte
Pentium	4kbyte - 4Mbyte
Power PC	4kbyte

주어 진 크기의 TLB에 대하여 프로쎄스의 기억기크기가 증가함에 따라 그리고 국소성이 감소함에 따라 TLB에 관한 명중률은 한계값에 접근한다. 이런 상태에서 TLB는 성능의 병목으로 될수 있다(실례: [CHEN92]를 보시오.).

TLB성능을 개선하는 한가지 방도는 더 많은 입구점들을 가지는 보다 큰 TLB를 사용하는것이다. 그러나 TLB의 크기는 주기억기캐쉬와 명령주기당 기억기접근수와 같은 하드웨어설계의 다른 측면들과 호상 제약한다[TALL92]. 약점은 TLB의 크기가 주기억기의 크기만큼 빨리 증가되지 않는다는것이다. 보다 큰 페지크기를 사용하는 방안을 선택하여 TLB에서 매개 페지표입구점이 보다 큰 기억기의 블로크를 참조하도록 해야 한다. 그러나 큰 페지크기를 사용하면 성능저하를 일으킬수 있다는것을 방금 보았다.

따라서 많은 설계자들이 여러가지의 페지크기를 사용하는 문제들[TALL92, KHAL93]과 MIPS R4000, Alpha, UltraSPARC 및 Pentium 을 포함하여 여러가지의 페지크기를 지원하는 몇가지 극소형처리기구성방식들을 연구하였다. 여러가지의 페지크기는 TLB를 효과적으로 사용하는데 필요한 유연성을 보장한다. 실례를 들면 프로그람명령과 같은 프로쎄스의 주소공간에서 큰 련속구역을 많은 개수의 작은 페지가 아니라 작은 개수의 큰 페지를 사용하여 사영할수 있다. 그러나 가장 상업적인 조작체계들은 여

[3] 파라메터 W는 제8장 제2절에서 설명하는 개념인 작업모임의 크기를 보여 주고 있다.

전히 기초로 되고 있는 하드웨어의 능력을 고려하지 않고 하나의 페지크기만을 지원하고 있다. 그 리유는 페지크기가 조작체계의 많은 측면들에 영향을 준다는것인데 이로부터 여러개의 페지크기로 변화시킨다는것은 복잡한 문제로 되고 있다(설명은 [GANA98]을 보시오.).

토막화

가상기억기의 본질

토막화는 프로그람작성자가 기억기를 다중주소공간 즉 토막들로 구성되여 있는것으로 보게 한다. 토막들은 서로 같지 않는 실제상 동적인 크기로 될수 있다. 기억기참조는 주소의 형태(토막번호, 편위)로 구성된다.

이 조직은 비토막식주소공간에 비해 프로그람작성자에게 많은 유리한 점을 준다. 즉

1. 늘어 나는 자료구조의 처리를 간소화시킨다. 만일 프로그람작성자가 특정한 자료구조가 얼마나 커지겠는지 사전에 알지 못한다면 동적토막크기가 허용되지 않는한 추측해야 한다. 토막식가상주소를 사용하면 자료구조를 그자체의 토막에 할당할수 있으며 조작체계는 필요에 따라 토막을 확대하거나 축소한다. 만일 확대되는데 필요한 토막이 주기억기에 있는데 공간이 불충분하다면 조작체계가 가능하다면 그 토막을 주기억기의 더 큰 령역으로 이동시킬수도 있고 또는 그것을 교체하여 내 보낼수도 있다. 후자의 경우 확대된 토막은 다음 기회에 교체되여 들어 온다.
2. 프로그람의 전체적인 모임을 요구함이 없이 독립적으로 변경하고 재콤파일하여 다시 련결하고 다시 적재할수 있다. 또한 이것은 다중토막들을 사용하여 수행된다.
3. 그자체를 프로쎄스들사이에서 공유하도록 한다. 프로그람작성자는 편의프로그람이나 유용한 자료표를 다른 프로쎄스가 참조할수 있는 어떤 토막에 배치할수 있다.
4. 그자체가 보호에 기여하도록 한다. 토막을 잘 정의된 프로그람이나 자료의 모임을 포함하도록 구성할수 있기때문에 프로그람작성자나 체계관리자가 편리한 방식으로 접근권한을 할당할수 있다.

구성

단순토막화에 대한 설명에서는 매개 프로쎄스가 자기자체의 토막표를 가지며 그 토막모두가 주기억기에 적재될 때 프로쎄스에서의 토막표가 창조되여 주기억기에 적재된다는것을 지적하였다. 매개 토막표입구점은 주기억기에서 대응하는 토막의 시작주소는 물론 그 토막의 길이도 가지고 있다. 토막화에 기초한 가상기억기방안을 고찰할 때 같은 장치, 토막표가 요구된다. 또 대체로 단일토막표를 매개 프로쎄스와 련관시킨다. 그러나 이 경우에 토막표입구점들은 더 복잡해 진다(그림 8-2 ㄴ). 프로쎄스의 일부 토막들만이 주기억기에 있을수 있으므로 대응하는 토막이 주기억기에 있는가 없는가를 지적하기 위하여 매개 토막의 입구표에 한개의 비트가 있어야 한다. 그 비트가 토막이 기억기에 있다는것을 지적해 주면 입구점은 또한 시작주소와 그 토막의 길이를 포함한다.

토막의 입구표에 필요한 또 다른 조종비트는 변경비트인데 대응하는 토막의 내용이 그 토막이 마지막으로 주기억기에 적재된 때로부터 변경되였는가 아닌가를 지적한다. 아무러한 변화도 없었다면 그것이 현재 차지하고 있는 프레임에서 토막을 치환할 시간이

울 때 그 토막을 외부쓰기할 필요가 없다. 다른 조종비트를 또 줄수도 있다. 실례로 보호나 공유를 토막준위에서 관리한다면 그 목적을 위한 비트들이 필요할것이다.

기억기에서 단어를 읽는 기본수법은 토막번호와 편위로 구성되여 있는 가상 즉 론리주소를 토막표를 사용하는 물리적주소에로 변환하는것을 동반한다. 토막길이가 프로쎄스의 크기에 따라 길이가 변하기때문에 그것을 등록기들에 유지하게는 할수 없다. 대신 그것에 접근할수 있도록 주기억기에 있어야 한다. 그림 8-12 는 이 방안에 대한 하드웨어실현을 제의하고 있다. 특정한 프로쎄스가 실행하고 있을 때 등록기는 프로쎄스에 대한 토막표의 시작주소를 보존한다. 가상주소토막번호는 표를 첨수주소화하며 그 토막의 시작에 대한 주기억기의 주소를 찾아 보는데 사용한다. 이것을 가상주소의 편위부분에 더하여 요구하는 실제주소를 산생시킨다.

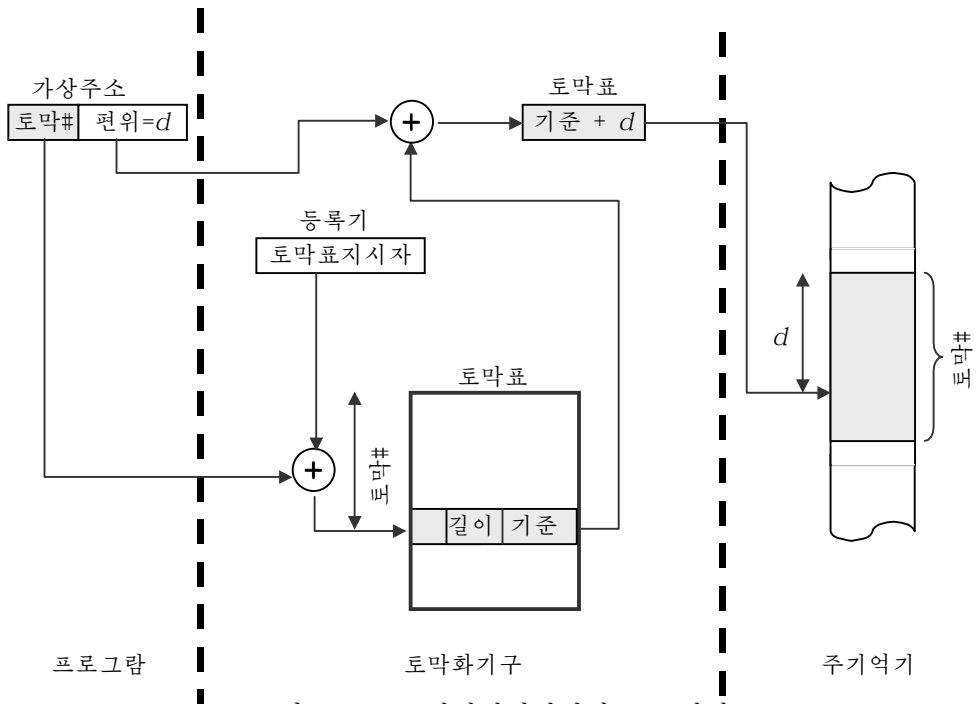



그림 8-12. 토막화체계에서의 주소변환

조합식페지화와 토막화

페지화와 토막화는 둘 다 자기들의 위력을 가지고 있다. 프로그람작성자에게 투명한 페지화는 외부쪼각을 제거함으로써 주기억기를 효과적으로 사용할수 있게 한다. 게다가 주기억기의 안팎으로 이동하는 쪼각들이 고정된 같은 크기로 되여 있으므로 앞으로 보게 될 프로그람들의 성질을 조종하는 정교한 기억기관리알고리듬을 개발할수 있게 한다. 프로그람작성자가 변경할수 있는 토막화는 앞에서 지적한 위력을 가지며 늘어 나는 자료구조, 모듈성을 조종하고 공유 및 보호를 지원하는 능력을 가진다. 량자의 우점을 결합시키기 위하여 일부 체계들에서는 처리기의 하드웨어와 조작체계쏘프트웨어로 장비하여 량자의 우점을 살리고 있다.

조합식페지화/토막화체계에서 사용자의 주소공간은 프로그람작성자가 마음대로 몇개의 토막들로 분할한다. 매개 토막들은 또 몇개의 고정된 크기의 페지들로 분할되는데 그

것은 주기억기프레임과 길이가 같다. 프로그람작성자의 견지에서 론리적주소는 여전히 토막번호와 토막편위로 구성되여 있다. 체계의 견지에서 토막편위를 특정한 토막안에 있는 페지에서의 페지번호와 페지편위로 본다.

그림 8-13에서는 조합식페지화/토막화를 지원하는 구조를 제기하고 있다. 매개 프로쎄스와 관련된것은 한개의 토막표와 많은 페지표인데 프로쎄스토막당 하나씩 있다. 특정한 프로쎄스가 실행하고 있을 때 등록기는 프로쎄스의 토막표시작주소를 보존하고 있다. 가상주소로 표현한다면 처리기는 토막번호부분을 사용하여 프로쎄스의 토막표를 첨수주소화하고 토막화에서의 페지표를 찾아 낸다. 그다음 가상주소의 페지번호부분을 사용하여 페지표를 첨수주소화하고 대응하는 프레임번호를 찾아 본다. 이것이 가상주소의 편위부와 조합되여 요구하는 실제주소를 산생시킨다.

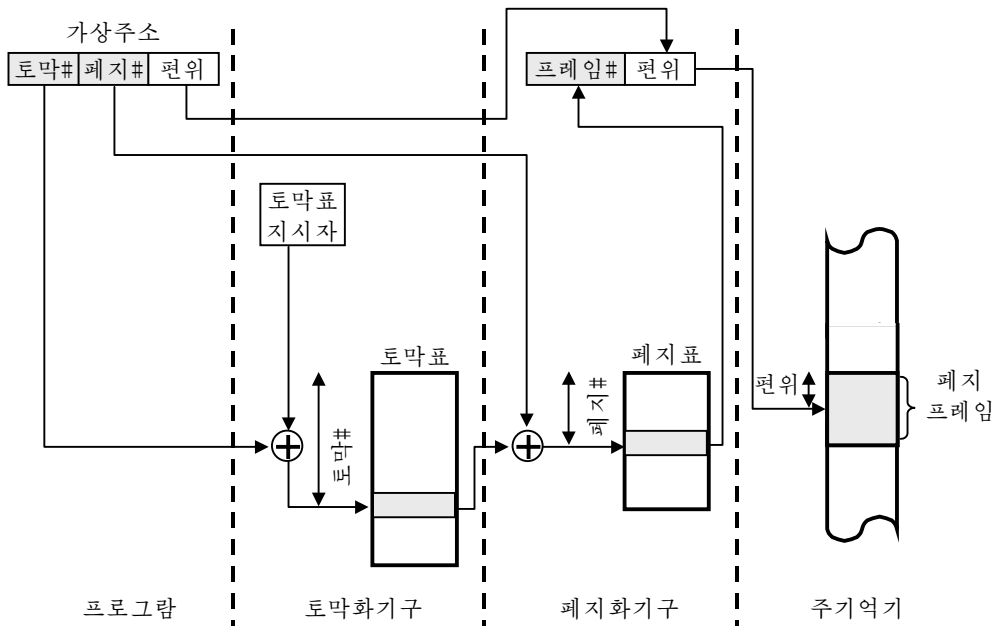



그림 8-13. 토막화/페지화체계에서의 주소변환

그림 8-2 ㄷ는 토막표입구점과 페지표입구점의 양식을 제기하고 있다. 이전과 같이 토막표입구점은 토막의 길이를 포함하고 있다. 그것은 또한 기준마당을 포함하고 있는데 이것을 이제부터 페지표라고 한다. 제시 및 변경 비트들이 필요 없는데 그것은 이것들이 페지준위에서 처리되기때문이다. 공유 및 보호를 목적으로 다른 조종비트를 사용할수도 있다. 페지표입구점은 본질적으로 순수한 페지화체계에서 사용되는것과 같은것이다. 매개 페지번호는 페지가 주기억기에 있다면 대응하는 프레임번호로 사영된다. 변경된 비트는 프레임이 또다른 페지에 배정될 때 페지를 거꾸로 외부쓰기해야 하는가 아닌가를 지적한다. 보호나 기억기관리의 다른 측면들을 처리하는 다른 조종비트들이 있을수도 있다.

보호와 공유

토막화는 그자체가 보호 및 공유방책에 기여한다. 매개 토막표입구점이 길이는 물론 기준주소를 가지고 있기때문에 어떤 프로그람이 토막의 한계를 넘어 주기억기위치에 예

고 없이 접근할수 없다. 공유를 실현하면 토막이 하나이상의 프로쎄스의 토막표들에서 참조될수 있다. 물론 같은 수법을 페지화체계에서 사용할수 있다. 그러나 이 경우에 프로그람과 자료의 페지주소는 프로그람작성자에게 보이지 않으며 보호 및 공유의 요구사항들을 더 다루기 힘들게 만든다. 그림 8-14 는 그러한 체계들에서 실행될수 있는 보호관계의 형태들을 설명하고 있다.

보다 정교한 수법을 적용할수도 있다. 공통적인 방안은 제 3 장에서 언급한 형태인 고리형보호구조를 사용하는것이다(련습문제 3-16). 이 방안에서 보다 낮은 번호를 가진 즉 보다 안쪽에 있는 고리들은 높은 번호를 가진 즉 바깥쪽 고리들보다 더 큰 권한을 발휘할수 있다. 대체로 고리 0 은 보다 높은 준위에 있는 응용프로그람인것으로 하여 조작체계의 핵심부기능들에 예약되여 있다. 일부 편의프로그람이나 조작체계봉사프로그람들은 중간고리를 차지할수 있다. 고리체계의 기본원리는 다음과 같다. 즉

1. 프로그람은 같은 고리 또는 보다 낮은 권한을 가진 고리에 있는 자료에만 접근할수 있다.
2. 프로그람은 같거나 더 많은 권한을 가진 고리에 있는 봉사들을 호출할수 있다.

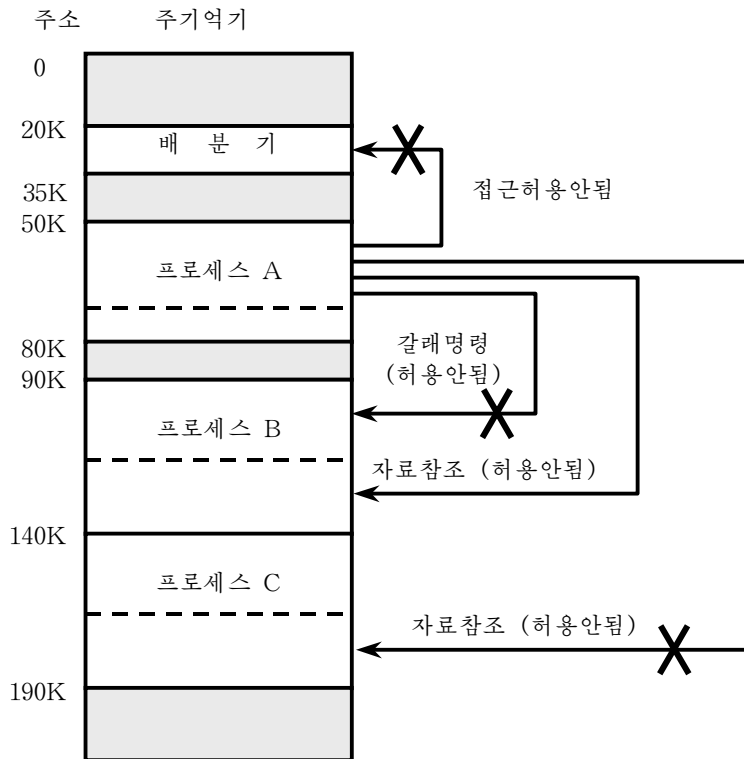



그림 8-14. 토막들사이에서의 보호관계

제 2 절. 조작체계의 쏘프트웨어

조작체계에서 기억기관리부분의 설계는 세가지 기본 령역의 선택에 관계된다. 즉

- 가상기억기수법을 사용하는가 안하는가
- 페지화 또는 토막화 아니면 둘 다 사용하는가
- 기억기관리의 여러가지 측면에서 사용되고 있는 알고리듬

첫 두개 령역에서의 선택은 사용할수 있는 하드웨어가동환경에 관계된다. 그래서 초기의 UNIX 실현물들은 체계를 실행한 처리기들이 페지화나 토막화를 지원하지 못했기때문에 가상기억기를 제공하지 못하였다. 이 수법들중에서 어느것이든 주소변환 및 다른 기본 기능들에 대한 하드웨어적지원이 없으면 실천적인것이 못된다.

우에서 지적한 첫 두개의 항목들에 대하여 두가지를 보충적으로 해석한다. 즉 우선 MS-DOS 와 같은 보다 낡은 일부 개인용콤퓨터에서의 조작체계와 전문화된 체계들은 제외하고 모든 중요한 조작체계들이 가상기억기를 보장한다. 둘째로, 순수한 토막화체계는 점차 적용하지 않고 있다. 토막화를 페지화와 조합할 때 조작체계설계자들이 부닥치게 되는 기억기관리문제는 페지화의 령역이다.[4] 그러므로 이 절에서는 페지화와 관련되여 있는 문제들에 집중한다.

셋째 항목과 관련되는 선택은 조작체계쏘프트웨어의 분야이며 이 절의 주제로 된다. 표 8-3 에서는 고려하는 주요한 설계요소들을 목록화하고 있다. 매 경우에 기본 문제는 성능 한가지이다. 즉 페지부재가 발생하는 비률을 최소화하는것인데 그것은 페지부재가 상당한 쏘프트웨어의 간접소비시간을 가져 오기때문이다. 최소한 간접소비시간에는 어느 상주페지 또는 페지들을 치환하겠는가를 결정하는 문제와 페지를 교환하는데서의 입출력이 포함된다. 또한 조작체계는 다른 프로쎄스를 일정작성하여 프로쎄스를 절환시키는 페지의 입출력기간에 실행시켜야 한다. 따라서 프로쎄스가 집행하고 있는 시간동안에 빠진 페지상에 있는 단어를 참조할 가능성이 최소가 되도록 일감을 배치하여야 한다. 표 8-3 에서 언급된 모든 령역들에는 가장 좋게 작업시키는 그 어떤 정해진 방책도 없다. 보는것처럼 페지화환경에서 기억기관리과제는 매우 복잡하다. 더우기 임의의 특정한 방책들의 성능은 주기억기의 크기, 주 및 2 차기억기의 상대적속도, 자원을 놓고 경쟁하는 프로쎄스들의 크기와 수 그리고 개별적프로그람들의 집행동작에 의존하고 있다. 이 후자의 특성은 응용프로그람, 사용되는 프로그람작성언어와 콤파일러, 그것을 작성한 프로그람작성자의 문제에 관계되며 또 대화형프로그람에서는 사용자의 동적인 행동에도 관계된다. 그러므로 여기서나 임의의 곳에서 그 어떤 최종적인 대답도 기대하지 말아야 한다. 보다 규모가 작은 체계에서 조작체계설계자는 현재의 지식상태에 기초하여 넓은 령역의 조건에 관하여 《좋다》고 보이는 방책들의 모임을 선정하여야 한다. 보다 규모가 큰 체계 특히 일반형콤퓨터인 경우에 조작체계는 싸이트관리자가 조작체계를 조화시켜 싸이트의 조건들에 기초한 《좋은》결과들을 얻을수 있게 해 주는 감시 및 조종도구를 갖추어야 한다.

---

[4] 보호와 공유는 보통 조합식토막화/페지화체계의 토막준위에서 취급한다. 이 문제는 다음 장들에서 취급한다.

표 8-3. 가상기억기의 조작체계방책

불러내기방책	**상주모임관리**
요구	상주모임크기
미리페지화	고정됨
배치방책	가변적임
	치환범위
치환방책	전역
기본알고리듬	국부
최적	**지우기방책**
최대미사용(LRU)	요구
선입선출(FIFO)	미리지우기
시계	**적재조종**
페지완충	다중프로그람작성정도

불러내기방책

불러내기방책은 언제 페지를 주기억기에 가져다 넣겠는가를 결정한다. 두가지 공통적인 선택안은 요구페지화 및 미리페지화이다. **요구페지화**에서는 페지상의 위치에 대하여 참조가 이루어질 때에만 페지를 주기억기에 가져다 넣는다. 기억기관리방책의 다른 요소들이 만족된다면 다음과 같은 현상들이 일어 날수도 있다. 프로쎄스를 처음으로 시작할 때 페지부재혼란이 일어 난다. 페지를 점점 더 많이 가져다 넣을수록 국소성의 원리로부터 후에 대부분의 참조가 최근에 가져다 넣은 페지에서 이루어 진다. 그러므로 어떤 시간후에는 문제거리가 평정되여 페지부재의 수는 매우 낮은 수준까지 떨어 진다.

미리페지화에서는 페지부재로부터 요구되는 페지가 아닌 다른 페지를 가져다 넣는다. 미리페지화는 자리찾기시간과 회전지연시간을 가지는 디스크와 같은 대부분의 2 차기억기의 특성들을 리용한다. 이것은 만일 프로쎄스의 페지들이 2 차기억기에서 린접하여 기억되여 있다면 이따금 그것들을 끌어 들이는것보다 몇개의 린접한 페지들을 한번에 끌어다 넣는것이 더 효과적이다. 물론 끌어 들인 대부분의 나머지 페지들이 참조되지 못한다면 이 방책은 효과적이지 못하다.

프로쎄스가 첫 시작을 할 때에는 어쨌든 프로그람작성자가 요구하는 페지를 지명해야 한다. 또는 매번 페지부재가 발생할 때 미리페지화방책을 사용할수 있다. 후자의 과정은 그것이 프로그람작성자에게 보이지 않으므로 더 좋은듯하다. 그러나 미리페지화의 실용성은 확정되여 있지 않다[MAEK87].

미리페지화를 페지교체와 혼동하지 말아야 한다. 페지교체에서는 프로쎄스가 기억기밖으로 교체되여 나가 중단된 상태에 놓일 때 모든 상주페지들이 이동되여 나간다. 프로쎄스를 재개할 때 이미 주기억기에 있던 모든 페지들은 주기억기에로 다시 들어 간다.

배치방책

배치방책은 실제기억기에서 프로쎄스쪼각이 어디에 상주하겠는가를 판정한다. 순수토막화체계에서는 배치방책이 중요한 설계문제로 되는데 제 7 장에서 설명한 최적적합방책, 최초적합방책 등과 같은 방책들을 선택할수 있다. 그러나 순수한 페지화든지 토막화와 조합된 페지화를 사용하는 체계에서 배치는 보통 적합하지 않은데 그것은 주소변환하드웨어와 주기억기접근하드웨어가 임의의 페지프레임조합에서 자기들의 기능들을 동일한 효률을 가지고 수행할수 있기때문이다.

배치방책과 관련한 연구 및 개발이 계속 진행되고 있다. 이른바 비통일적인 기억기접근(NUMA)다중처리기상에서 기계의 분산형공유기억기를 기계상의 임의의 처리기가 참조할수 있으나 특정한 물리적위치에 접근하는 시간은 처리기와 기억기모듈사이의 거리에 따라 변한다. 그러므로 성능은 자료가 그것들을 사용하는 처리기에 얼마나 가까이 있는가에 크게 좌우된다[LARO92, BOLO89, COX89]. NUMA 체계에서는 자동배치전략이 페지들을 가장 좋은 성능을 주는 기억기모듈에 할당하는데서 적합하다.

치환방책

대부분의 조작체계에 관한 책들에서 기억기관리에 대한 처리는 《치환방책》이라는 표제밑에 새로운 페지를 끌어다 넣어야 할 때 기억기에서 페지를 선택하여 치환하는 문제를 취급한다. 이 문제는 때로는 설명하기가 힘든데 그것은 호상련관되여 있는 몇가지 개념들이 포함되기때문이다. 즉

- 매개 능동프로쎄스에 배정해야 할 페지프레임의 량
- 치환에서 고려해야 할 페지의 모임이 페지부재를 일으킨 프로쎄스에 국한되는지 아니면 주기억기의 모든 페지프레임을 다 포괄하는지
- 고려되는 페지모임중에서 치환에 선택되는 특정한 페지는 어느것인지

첫 두 개념을 상주모임관리라고 하는데 이것은 다음 부분절에서 취급하며 세번째 개념에 대해서는 치환방책이라는 용어를 붙여 이 부분절에서 설명한다.

치환방책의 분야는 아마 지난 20 여년에 걸쳐 기억기관리들에서 가장 많이 연구된 분야일것이다. 주기억기에서 모든 프레임들이 차지하고 새로운 페지를 끌어 들여 페지부재를 만족시켜야 할 때 치환방책은 기억기에서 현재 어느 페지를 치환하겠는가를 결정한다. 모든 방책들에서는 적어도 가까운 장래에 참조될수 있음직한 페지로 될 바로 그 페지를 자기의 대상으로 한다. 국소성의 원리로 하여 최근 참조경력과 가까운 장래의 참조패턴들사이에는 흔히 높은 상관이 있다. 그러므로 대부분의 방책들은 과거의 동작에 기초하여 가까운 장래의 동작을 예측한다. 고려해야 할 한가지 절충방안은 치환방책이 정교하면 할수록 그것을 실현하기 위한 하드웨어와 쏘프트웨어의 간접소비시간은 점점 더 커진다는것이다.

프레임페쇄법

각이한 알고리듬들을 설명하기전에 치환방책에 관한 한가지 제약조건을 언급할 필요가 있다. 즉 주기억기에서 일부 프레임들은 페쇄시킬수도 있다. 프레임이 페쇄될 때 현재 프레임에 기억된 페지는 치환할수 없다. 조작체계의 많은 핵심부는 페쇄된 프레임들상에 보존될뿐아니라 물론 기본 조종구조로 된다. 그밖에 입출력완충기들과 다른 시간림계구역들을 주기억기프레임들로 페쇄시킬수 있다. 페쇄법은 페쇄비트를 매개 프레임과 련관시켜 실현할수 있다. 물론 이 비트를 현재 페지표에 포함되여 있는 프레임표에 보관할수 있다.

기본알고리듬

상주모임관리전략에 관계없이(다음의 소절에서 설명한다.) 치환할 페지선택에 사용되는 일정한 기본 알고리듬들이 있다. 문헌들에서 고찰하여 온 치환알고리듬들로는 다음과 같은것들이 있다. 즉

- 최적알고리듬
- 최대미사용(LRU)알고리듬
- 선입선출(FIFO)알고리듬
- 시계알고리듬

최적방책은 다음 참조시간이 가장 긴 페지를 치환용으로 선택한다. 이 알고리듬이 가장 적은 수의 페지부재를 초래한다는것을 보여 준다[BELA66]. 명백히 이 알고리듬은 실현불가능한데 그것은 조작체계가 장래의 사건들을 완전히 확증해야 하기때문이다. 그러나 이 알고리듬은 다른 알고리듬들을 판정하는 기준으로 봉사한다.

그림 8-15에서는 최적방책에 대한 실례를 주고 있다. 이 실례는 세개의 프레임중에서 이 프로쎄스에 대한 고정된 프레임배정(고정된 상주모임크기)을 가정하고 있다. 프로그람의 집행으로 형성되는 페지주소흐름은

2 3 2 1 5 2 4 5 3 2 5 2

인데 이것은 참조된 첫 페지가 2 이고 두번째 참조된 페지가 3 …이라는것을 의미한다. 최적방책은 프레임배정이 끝난다음에 세개의 페지부재를 산생시킨다.

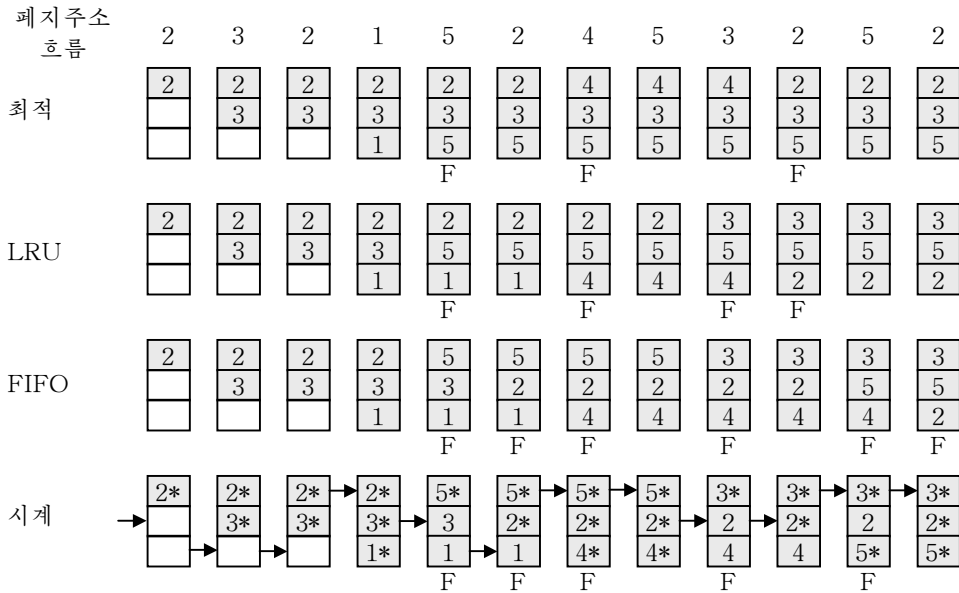



그림 8-15. 네개의 페지치환알고리듬의 동작

최대미사용(LRU)방책은 가장 오랜 시간동안 참조되지 않는 기억기안의 페지를 치환한다. 국소성의 원리에 의해 이것은 가까운 장래에 가장 적게 참조됨직한 페지로 된다. 그리고 사실상 LRU 방책은 거의나 최적인 방책으로 된다. 이 방법과 관련한 문제는 실현하기가 곤난한것이다. 한가지 방법은 그의 마지막 참조시간을 가지는 매개 페지에 태그를 달아 주는 방법인데 이것은 모든 명령과 자료를 기억기참조할 때마다 진행하여야 한다. 하드웨어가 그러한 방안을 지원한다고 해도 간접소비시간은 굉장히 커진다. 대신 페지참조용으로 탄창을 사용할수 있는데 이것역시 값이 많이 든다.

그림 8-15에서는 LRU 의 동작에 대한 실례를 보여 주고 있는데 최적방책실례에서와 같은 페지주소흐름을 사용하고 있다.

선입선출방책은 프로쎄스에 배정된 페지프레임들을 순환완충기로 처리하며 페지들은 순환식으로 제거한다. 요구되는것은 모두 프로쎄스의 페지프레임들을 원을 짓게 해 주는 지시기이다. 따라서 이것은 가장 간단한 페지치환방책중의 하나가 실현되는것으로 된다. 이 선택을 뒤받침하는 론리는 그것의 단순성에 있는것이 아니라 주기억기에 가장 오래 있은 페지를 치환한다는 바로 그것이다. 즉 오래전에 기억기에 불러 들인 페지가 이제는 사용에서 떨어 져 나갈수 있다는것이다. 이러한 론의는 흔히 잘 맞지 않는데 그 리유는 프로그람의 수명전기간에 걸쳐 많이 사용되는 프로그람이나 자료구역이 자주 존재하기때문이다. 그러한 페지들이 선입선출알고리듬에 의해 자주 페지화되여 페지넣기 및 페지출구된다.

그림 8-15에서의 실례를 보면 FIFO 방책이 6개의 페지부재를 일으킨다. LRU 는 2페지와 5페지가 다른 페지들보다 더 자주 참조되는 반면에 FIFO 는 그렇지 못하다는데 주목하자.

LRU방책이 거의 최적인 방책으로서 잘 동작하는 반면에 실현하기 힘들고 현저한 간접소비시간을 부과하기가 힘들다. 다른 한편 FIFO방책은 매우 간단하여 실현할수 있으나 상대적으로 약하게 동작한다. 여러해동안 조작체계설계자들은 간접소비시간을 적게 하면서 LRU의 성능에 가까운 많은 다른 알고리듬들을 시도하였다. 이 많은 알고리듬들은 **시계방책**이라고 하는 방안의 변종들이다.

시계방책의 가장 간단한 형태는 사용비트라고 하는 보충적인 비트를 매개 프레임과 련관시킬것을 요구한다. 페지가 기억기의 프레임에 처음으로 적재될 때 프레임에 대한 사용비트가 1 로 설정된다. 페지가 계속 참조될 때(페지부재가 발생한 참조후에) 그의 사용비트는 1 로 설정된다. 페지의 치환알고리듬에서 치환후보로 되는 프레임모임(프로쎄스: 국부유효범위: 모든 주기억기: 전역유효범위 [5])은 지시자가 관련되는 순환완충기로 고찰한다. 페지가 치환될 때 지시자는 완충기의 다음 프레임을 가리키기 위하여 설정된다. 페지를 치환할 시간이 올 때 조작체계는 완충기를 주사하여 사용비트가 0 으로 설정되여 있는 프레임을 찾는다. 사용비트가 1 인 프레임과 맞다 들릴 때마다 비트를 0 으로 재설정한다. 만일 완충기에서 임의의 프레임이 이 처리의 초기에 사용비트를 0 으로 가지고 있다면 맞다 들린 첫 프레임이 치환용으로 선택된다. 만일 모든 프레임들이 사용비트를 1 로 가진다면 지시자는 완충기를 통한 하나의 완전한 주기를 만들며 모든 사용비트를 0 으로 설정하여 초기위치에서 정지하며 프레임의 페지를 치환한다. 시계방책에서 사용비트를 1 로 가지는 임의의 프레임의 알고리듬으로 넘어 가는것을 제외하고는 이 방책이 FIFO 와 류사하다는것을 알수 있다. 그 방책을 시계방책이라고 하는것은 페지프레임들을 원에 펼쳐 놓은것처럼 볼수 있기때문이다. 많은 조작체계들에서는 이 간단한 시계방책의 일부 변종들을 사용하고 있다(실례로 Multics [CORB68]).

그림 8-16에서는 간단한 시계방책수법에 대한 실례를 보여 주고 있다. n-1개의 주기억기프레임들로 된 순환완충기는 페지치환에서 사용할수 있다. 727페지가 들어 오는것과 관련하여 완충기에서 어떤 페지를 치환하기전에 다음 프레임지시자는 45페지를 포함하고 있는 2프레임을 지적한다. 시계방책은 이제부터 집행된다. 2프레임의 45페지에 대한 사용비트는 1과 같기때문에 이 페지는 교체되지 않는다. 대신에 사용비트는 0으로 설정되며 지시자는 전진한다. 류사하게 3프레임의 191페지가 치환되지 않는데 사용비트는 0으로 설정된다. 따라서 556페지는 727페지와 교체된다. 사용비트는 프레임에 대해 1로 설정되며 지시자는 5프레임에로 전진하여 페지의 치환수속을 끝낸다.

---

[5] 유효범위에 대한 개념은 소절《치환의 유효범위》에서 계속 론의한다.

시계방책의 동작을 그림 8-15 에서 설명한다. 별표가 있는것은 대응하는 사용비트가 1 과 갈다는것을 가리키며 화살표는 현재 지시자의 위치를 가리킨다. 시계방책은 2 프레임과 5 프레임을 치환으로부터 보호하는데 쓰인다.

그림 8-17에서는 [BAER80]에서 보고된 실험의 결과를 보여 주고 있는데 이것은 론의한 4개의 알고리듬을 비교하고 있으며 프로쎄스에 할당된 몇개의 페지가 고정된다는것을 가정하고 있다. 결과들은 FORTRAN 프로그람에서 0.25×10^6참조의 집행에 기초하고 있으며 페지크기는 256단어를 사용하고 있다. Baer 는 6, 8, 10, 12 및 14개의 프레임배정에 관하여 실험을 하였다. 네가지 방책들사이에서 차이는 배정이 작을 때 가장 치명적인데 FIFO 는 최적방책인 경우보다 2배이상 더 나쁘다. [FINK88]에서도 거의동일한 결과가 보고되였는데 역시 최대퍼짐이 대략 2배라는것을 보여 주고 있다. Finkel 의 방법은 100개 페지의 가상공간에서 선택된 10000개의 합성된 페지참조렬에 대한 각이한 방책들의 효과를 모의하는것이였다. 국소성원리의 효과를 근사화하기 위하여 특정한 페지를 참조할 확률로서 지수분포를 적용하였다. Finkel 은 축에서 단지 2배일 때 정교한 페지치환알고리듬들에서 거의 문제가 없다는 결론을 내릴수 있었다. 그러나 이 차이가 주기억기의 요구에 대해서든(조작체계의 성능저하를 피하기 위하여) 조작체계의 성능에 대해서든(주기억기가 확대되는것을 피하기 위하여) 현저한 효과를 가지게 된다는것을 지적하고 있다.

시계알고리듬은 또한 가변배정 및 전역치환범위든 국부치환범위든(치환방책의 다음의 설명을 보시오.) 사용될 때[CARR81, CARR84] 다른 알고리듬들과 비교하였다.
결과 시계알고리듬들이 LRU 의 성능에 거의 가까이 접근한다는것을 알게 되였다.

그것이 사용하는 비트수를 증가시켜 시계알고리듬을 더 강력하게 만들수 있다.[6] 페지화를 지원하는 모든 처리기들에서 변경비트는 주기억기의 매개 페지와 련관되여 있으므로 주기억기의 매개 프레임과도 련관되여 있다. 이 비트는 페지가 변경되였을 때 그것이 2 차기억기에 씌여질 때까지 치환되지 않도록 하기 위해 필요하다. 시계알고리듬에서 이 비트를 다음과 같은 방법으로 사용할수 있다. 만일 사용자변경비트를 고려한다면 매개 프레임은 4 개의 범위중의 하나에 속한다. 즉

- 최근에 접근되지 않음, 변경되지 않음($u = 0$; $m = 0$)
- 최근에 접근됨, 변경되지 않음($u = 1$; $m = 0$)
- 최근에 접근되지 않음, 변경됨($u = 0$; $m = 1$)
- 최근에 접근됨, 변경됨($u = 1$; $m = 1$)

이 분류에 따라 시계알고리듬은 다음과 같이 동작한다. 즉

1. 지시자의 현 위치에서 시작하여 프레임완충기를 주사한다. 이 주사기간 사용비트에 아무런 변화도 주지 않는다. ($u = 0$; $m = 0$)에 관하여 맞다 들린 첫 프레임을 치환용으로 선택한다.
2. 1 단계가 실패하면 다시 주사하여 ($u = 0$; $m = 1$)을 가진 프레임을 찾는다. 처음 맞다 들린 프로그람을 치환용으로 선택한다. 이 주사기간에 통과시킨 매개 프레임에 대해 사용비트를 0 으로 설정한다.
3. 2 단계가 실패하면 지시자는 본래 위치에로 복귀되며 모임안의 모든 프레임들은 사용비트를 0 으로 가진다. 1 단계를 반복하며 필요하면 2 단계를 반복한다. 이때 어떤 프레임이 치환용으로 발견된다.

---

[6] 다른 한편으로 사용되는 비트수를 령으로 줄이면 시계알고리듬은 FIFO 로 퇴화된다.

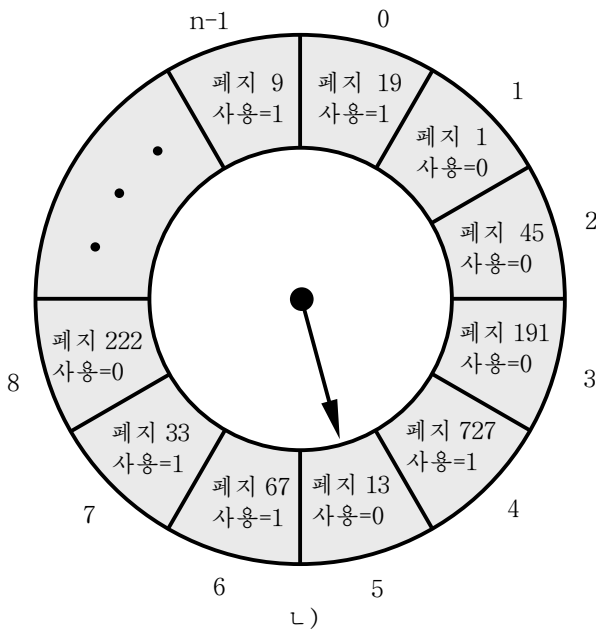
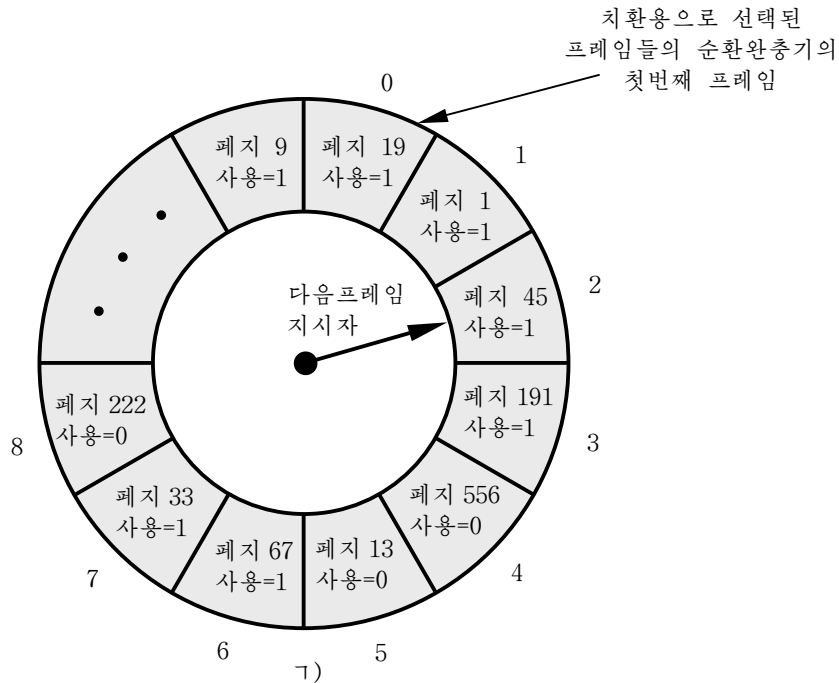



그림 8-16. 시계방책의 조작실례
ㄱ-페지치환직전의 완충기의 상태, ㄴ-다음페지의 치환직후에 완충기의 상태

요약하여 말한다면 페지치환알고리듬은 완충기의 모든 페지들을 순환하면서 끌어들인후에 변경되지 않은것과 최근에 접근을 받지 않은것을 찾는다. 그러한 페지가 치환의 좋은 대상으로 되며 그것이 변경되지 않기때문에 2 차기억기에 도로 쓰기할 필요가 없다는 우점을 가지고 있다. 첫 훑기에서 아무런 후보페지도 발견되지 않으면 알고리듬은 완

충기를 다시 순회하여 최근에 접근되지 않은 변경된 페지를 찾는다. 그러한 페지를 분명 외부쓰기하여 치환해야 하지만 국소성의 원리로부터 그것은 임의의 시간에 곧 필요하지 않을수도 있다. 두번째 과정이 실패하면 완충기의 모든 프레임들에는 최근에 접근되지 않았다는 표시를 하고 세번째 훑기를 수행한다.

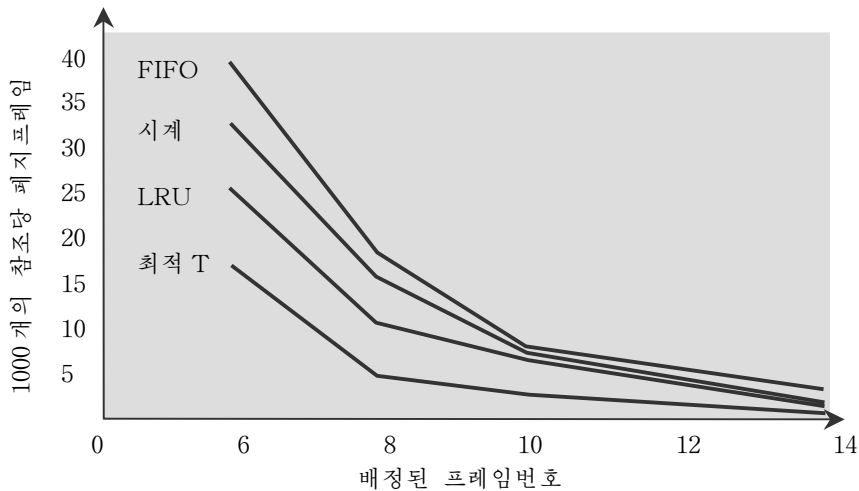



그림 8-17. 고정배정, 국부페지치환알고리듬의 비교

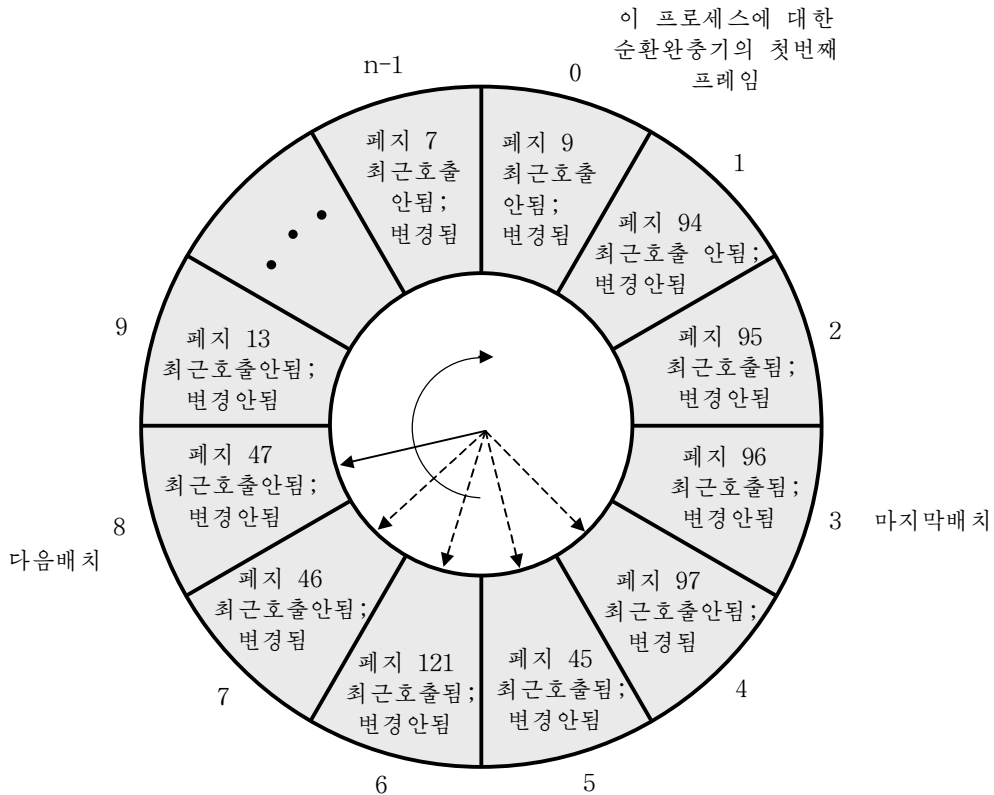



그림 8-18. 시계페지치환알고리듬[GOLD89]

이 전략은 Macintosh 가상기억기방안[GOLD89]에서 사용하고 있는데 그림 8-18 에서 설명하고 있다. 단순한 시계알고리듬에 비한 이 알고리듬의 우점은 변화되지 않은 페지가 교체용으로 선택된다는것이다. 변경된 페지를 치환되기전에 외부쓰기해야 하므로 직접 시간이 절약된다.

페지완충법

LRU 와 시계방책이 FIFO 보다 우월하다고 해도 그것들은 모두 FIFO 에서는 당하지 않는 복잡성과 간접소비시간을 포함한다. 그밖에 변경된 페지를 치환하는 비용이 전자는 2 차기억기를 외부쓰기해야 하기때문에 하지 않은 경우보다 더 커진다는 련관된 문제점이 있다.

페지화의 성능을 개선할수 있으며 더 간단한 페지의 치환방책을 사용할수 있는 흥미있는 전략은 페지완충법이다. VAX VMS 방법이 전형적이다. 페지의 치환알고리듬은 간단한 FIFO 이다. 성능을 개선하기 위해 치환된 페지를 없애는것이 아니라 두개 목록 즉 페지가 변경되지 않았다면 자유페지목록에 또는 변경되였다면 변경페지목록에 할당한다. 주목할것은 그 페지가 주기억기에서 물리적으로 이동한것이 아니라는것이다. 그대신이 페지에 대한 페지표의 입구점을 옮겨 자유 또는 변경된 페지목록중의 어느 하나에 배치한다.

자유페지목록은 페지들을 읽어 들이는데 사용할수 있는 페지프레임들의 목록이다. VMS 는 항상 일정한 작은 수의 자유로운 프레임들을 보존하려고 한다. 페지를 읽어 들일 때 목록의 맨 우에 있는 페지프레임을 사용하고 거기에 있던 페지를 파괴한다. 변경되지 않은 페지를 치환하려고 할 때 그것은 기억기에 남아 있으며 페지프레임은 자유페지목록의 꼬리에 추가된다. 류사하게 변경된 페지가 외부쓰기되여 치환하려고 할 때 페지프레임은 변경된 페지목록의 꼬리에 추가된다.

이 전술의 중요한 측면은 교체될 페지가 기억기에 남아 있다는것이다. 그러므로 프로쎄스가 페지를 참조한다면 그것은 적은 내용으로 프로쎄스의 상주모임에 반환된다. 사실상 자유로우며 변경된 페지목록은 페지들에 대한 캐쉬로 동작한다. 변경된 페지목록은 또다른 유용한 기능을 봉사해 준다. 즉 변경된 페지들은 한번에 한개가 아니라 클라스터들로 외부쓰기된다. 이것은 입출력조작의 수를 많이 줄이고 따라서 디스크접근시간의 량도 줄인다.

페지완충법의 보다 간단한 판본이 Mach 조작체계[RASH88]에서 실현되였다. 이 경우에 변경된 페지와 변경되지 않는 페지사이에는 아무런 구별도 없다.

치환방책과 캐쉬의 크기

이미 론의된바와 같이 주기억기의 크기는 점점 더 커지고 응용프로그람의 국소성은 감소하고 있다. 보충적으로 캐쉬크기가 증가하고 있다. 큰 캐쉬크기 지어 수메가바이트의 크기도 이제는 설계에서 선택할수 있게 되였다[BORG90]. 큰 캐쉬를 사용하여 가상기억기페지들을 치환하면 성능에서 효과를 거둘수 있다. 치환용으로 선택된 페지프레임이 캐쉬에 있으면 캐쉬블로크는 그것이 유지하는 페지와 함께 상실된다.

일정한 페지완충형태를 사용하는 체계에서 페지치환방책을 페지완충기에서의 페지방책과 함께 제공하여 캐쉬의 성능을 개선할수 있다. 대부분의 조작체계들은 페지완충기에서 독단적인 페지를 선택하여 페지들을 배치하는데 대체로 선입선출규칙을 사용한다.

[KESS92]에서 보고된 연구결과는 주의를 돌린 페지배치전략이 순수한 배치보다 캐쉬랑비가 10∼20% 더 적어 진다는것을 보여 주고 있다.

몇가지 페지배치알고리듬을 [KESS92]에서 고찰하고 있다. 세부적인것들은 이 책의 범위를 벗어 나는데 그것은 캐쉬구조와 방책들의 세부에 의존하기때문이다. 이 전략들의 본질은 같은 캐쉬블로크들에 사영되는 페지프레임의 수를 최소화하는 방법으로 주기억기에 련속적인 페지들을 끌어 들이는데 있다.

상주모임관리

상주모임의 크기

페지화된 가상기억기에서 집행을 위한 준비로서 프로쎄스의 모든 페지들을 주기억기에 끌어 들일 필요가 없으며 실제적으로 그렇게 할수도 없다. 그러므로 조작체계는 끌어들인 페지의 량 즉 특정한 프로쎄스에 배정할 주기억기의 량을 결정해야 한다. 이때 몇가지 인자들이 역할을 논다. 즉

- 프로쎄스에 배정된 기억기의 량이 작을수록 임의의 시각에 주기억기에 상주할수 있는 프로쎄스들은 더 많아 진다. 이것은 조작체계가 주어 진 임의의 순간에 적어도 한개의 준비프로쎄스를 찾아 낼 가능성을 증가시켜 주며 이로부터 교체로 인한 시간을 감소시킨다.
- 만일 상대적으로 적은 수의 프로쎄스의 페지가 주기억기에 있다면 국소성의 원리에도 불구하고 페지부재률이 오히려 높아 진다(그림 8-11 ㄴ를 보시오.).
- 일정한 크기이상 추가적인 주기억기를 특정한 프로쎄스에 배정하는것은 프로쎄스에서의 페지부재중에 눈에 띄울만한 아무러한 효과도 주지 못하는데 그것은 국소성의 원리때문이다.

이 인자들을 념두에 두고 조작체계들에서 적용하는 두가지 종류의 방책들을 고찰하려고 한다. **고정배정방책**은 프로쎄스에 주기억기의 고정된 수의 프레임들을 주어 그 안에서 실행시킨다. 그 수는 초기적재시간(프로쎄스의 창조시간)에 결정되며 프로쎄스의 형태(대화형, 일괄형, 응용프로그람의 형태)에 기초하여 결정될수 있으며 또는 프로그람작성자나 체계관리자의 안내에 기초할수도 있다. 고정배정방책에서는 프로쎄스의 집행시에 페지부재가 발생할 때마다 그 프로쎄스의 페지들중 한개를 요구되는 페지로 교체해야 한다.

가변배정방책은 프로쎄스에 배정된 몇개의 페지프레임을 프로쎄스의 수명기간에 가변시킬수 있다. 리상적으로 국소성의 원리가 그 프로쎄스에 대해 충분하지 못한 형식으로만 유지된다는것을 가리켜 주는 높은 수준의 페지부재를 지속적으로 받고 있는 프로쎄스는 추가적인 페지프레임들을 받아 페지부재률을 감소시킨다. 반면에 프로쎄스가 국소성의 견지에서 충분히 작용한다는것을 말해 주는 례외적으로 적은 페지부재률을 가진 프로쎄스는 이것이 페지부재률을 현저히 증가시키지 않을것이라는 기대를 반영하여 배정을 적게 받게 된다. 가변배정방책을 사용하는것은 다음 소절에 설명되여 있는바와 같이 치환범위에 대한 개념과 련관되여 있다.

가변배정방책은 보다 위력한 방책으로 될것이다. 그러나 이 방법에서의 난관은 조작체계가 능동프로쎄스들의 동작을 할당할것을 요구한다는것이다. 이것은 불가피하게 조작

체계에서 쏘프트웨어의 간접소비시간을 필요로 하며 처리기의 가동환경이 제공하는 하드웨어기구에 관계된다.

치환범위

치환전략의 범위를 전역 또는 국부범위로 분류할수 있다. 두 형태의 방책은 모두 아무런 자유페지프레임도 없을 때 페지부재에 의해 활성화된다. **국부치환방책**은 페지를 선택하여 치환하는데서 페지부재를 발생시킨 프로쎄스의 상주페지들속에서만 선택한다. **전역치환방책**은 프로쎄스가 특정한 페지를 소유하고 있는가 하는것을 고려하지 않고 주기억기의 페쇄되여 있지 않는 모든 페지들을 치환의 후보로 간주한다. 국부방책이 분석하기는 더 쉬운 반면에 전역방책보다 성능이 더 좋다는 아무러한 확고한 증거도 없지만 그것들은 실현의 간단성과 최소의 간접소비시간으로 해서 주목을 끌고 있다[CARR84, MAEK87].

표 8-4. 상주모임관리

	국부치환	**전역치환**
고정 배정	• 처리할 프레임의 수가 고정된다. • 치환될 페지가 그 프로쎄스에 배정된 프레임들로부터 선정된다.	• 불가능
가변 배정	• 프로쎄스에 배정된 프레임의 수를 시간에 따라 변화시켜 프로쎄스의 작업모임을 유지할수 있다. • 치환될 페지가 그 프로쎄스에 배정된 프레임들로부터 선정된다.	• 치환될 페지는 주기억기에서 사용할수 있는 모든 프레임들로부터 선정되는데 이것은 프로쎄스의 상주모임의 크기가 변할수 있게 해 준다.

치환범위와 상주모임크기사이에는 어떤 관계가 있다(표 8-4). 고정된 상주모임은 국부치환방책을 암시해 준다. 즉 고정된 상주모임의 크기를 유지하기 위해 주기억기에서 제거되는 페지를 동일한 프로쎄스로부터 다른 페지와 교체해야 한다. 가변배정방책은 명백히 전역치환방책을 사용할수 있다. 즉 주기억기의 한 프로쎄스로부터 페지를 다른 프로쎄스의 페지와 치환하는것은 한 프로쎄스의 배정을 한 페지씩 증가시키고 다른 프로쎄스의 배정을 한 페지씩 감소시킨다. 또한 가변배정과 국부치환이 유효한 조합이라는것을 알수 있다. 이제부터 이 세가지 조합을 고찰하자.

고정배정, 국부범위

이 경우에 고정된 수의 프레임을 가지고 주기억기에서 실행하고 있는 프로쎄스가 있다. 페지부재가 발생할 때 조작체계는 프로쎄스에 현재 있는 페지들중에서 치환할 페지를 선정하여야 한다. 이때 앞의 소절에서 론의한것과 같은 치환알고리듬을 사용할수 있다.

고정방책을 사용함에 있어서 배정시간을 미리 결정하여 프로쎄스에 주어야 한다. 응용프로그람의 형태와 프로그람이 요구하는 량에 기초하여 이것을 결정할수 있다. 이 방법의 결함은 2 중적이다. 즉 배정을 너무 작게 하면 페지부재률이 높아 져서 총적인 다중프로그람처리체계의 실행속도가 떨어 진다. 만일 배정을 불필요하게 크게 하면 주기억기에 너무 적은 프로그람들이 있어서 처리기의 휴식시간이 상당히 길어 진다든가 교체시 상당히 많은 시간을 소비하게 된다.

가변배정, 전역범위

이 조합은 실현하기가 가장 쉬워서 많은 조작체계들에서 받아 들이고 있다. 임의의 주어진 시간에 주기억기안에는 많은 프로쎄스들이 있으며 매개는 그에 배정된 일정한 수의 프레임들을 가지고 있다. 대체로 조작체계는 또한 자유프레임들에 대한 목록을 가지고 있다. 페지부재가 발생할 때 자유페지는 프로쎄스의 상주모임에 추가되여 그 페지를 끌어 들인다. 그러므로 페지부재에 부닥치는 프로쎄스는 점차 크기가 확대되는데 이것은 체계에서 전반적인 페지부재를 줄이게 한다.

이 방법에서의 난관은 치환의 선택에 있다. 사용할수 있는 자유페지가 전혀 없을 때 조작체계는 현재 주기억기에 있는 페지를 선정하여 교체해야 한다. 그 선택은 핵심부의 프레임이 같은 고정된 프레임을 제외하고 주기억기에 있는 모든 프레임들속에서 이루어진다. 앞의 소절에서 론의된 임의의 방책들을 사용하면 치환용으로 선택된 페지가 임의의 상주프로쎄스에 속할수 있는데 어느 프로쎄스가 그의 상주모임에서 페지를 잃겠는가를 판정하는 그 어떤 규칙도 없다. 따라서 상주모임의 크기가 작아 지는 영향을 받는 프로쎄스는 최적이 되지 못할수도 있다.

가변배정, 전역범위방책의 성능문제를 해결하는 한가지 방법은 페지완충법을 사용하는것이다. 이 방법에서 어느 페지를 치환하겠는가에 대한 선택은 의의가 적은데 그것은 페지의 어떤 블로크가 덧씌여 질 다음번 시간전에 참조된다면 그 페지를 다시 재생하여 리용할수 있기때문이다.

가변배정, 국부범위

가변배정, 국부범위전략은 전역범위전략과 관련한 문제점을 극복하기 위한것이다. 그것을 다음과 같이 요약할수 있다. 즉

1. 새로운 프로쎄스가 주기억기에 적재될 때 응용프로그람의 형태, 프로그람의 요청 또는 다른 기준에 근거하여 일정한 수의 페지프레임을 상주모임으로서 그것에 배정한다. 배정을 채우기 위하여 미리페지화 또는 요구페지화를 사용한다.
3. 페지부재가 발생할 때 부재의 영향을 받는 프로쎄스의 상주모임들속에서부터 치환하기 위한 페지를 선택한다.
4. 시간이 감에 따라 프로쎄스에 제공된 배정을 재평가하고 그것을 증가 또는 감소시켜 전체적인 성능을 개선한다.

이 전략에서 상주모임의 크기를 증가 또는 감소시키는 결정은 심중한것이며 그것은 응용프로쎄스에 대한 앞으로의 요구를 조사하는데 기초한다. 이 평가로 하여 그러한 전략은 간단한 전역치환방책보다 더 복잡하다. 그러나 그것이 더 좋은 성능을 낼수 있다.

가변배정, 국부범위전략의 기본요소는 상주모임의 크기와 변화의 시각을 결정하는데 쓰이는 기준이다. 문헌들에서 많은 주의를 돌린 하나의 특정한 전략은 **작업모임전략**으로 알려 져 있다. 실지의 작업모임전략이 실현하기 어려울수 있다 하여도 그것을 비교의 기준으로 고찰하는것은 쓸모가 있다.

작업모임은 Denning이 도입하고 대중화한 개념인데[DENN68, DENN70, DENN80b] 가상기억기관리설계에 깊은 충격을 주었다. 가상시간 t에서 어떤 프로쎄스에 대한 파라메터 Δ를 가지는 작업모임 $W(t, \Delta)$는 최종 Δ의 가상시간단위들에서 참조된 그

프로쎄스의 페지들의 모임이다. 가상시간에 대한 개념을 사용하여 프로쎄스가 실제적으로 집행하면서 경과한 시간을 표시한다. 가상시간을 명령주기의 측정단위로 생각할수 있는데 매개 집행된 명령은 하나의 시간단위와 같다.

W의 두개 변수들을 각각 고찰하여 보자. 변수 Δ는 프로쎄스를 관찰한 시간의 창문이다. 작업모임의 크기는 창문크기에 대한 비감소함수로 된다. 그 결과를 그림 8-19 에서 설명하고 있는데([BACH86]에 기초함) 이것은 프로쎄스에서의 페지참조의 순차를 보여 주고 있다. 점들은 작업모임이 변하지 않는 시간단위를 가리키고 있다. 창문크기가 커지면 커질수록 작업모임이 커진다. 이것을 다음과 같은 관계로 표현할수 있다. 즉

$$W(t,\Delta+1) \supseteq W(t,\Delta)$$

페지 참조의 순서	창문크기,△			
	2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	●	●
17	24 17	23 24 17	18 23 24	15 18 23 24
18	17 18	24 17 18	●	18 23 24 17
24	18 24	●	24 17 18	●
18	●	18 24	●	24 17 18
17	18 17	24 18 17	●	●
17	17	18 17	●	●
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	●
17	24 17	●	●	17 15 24
24	●	24 17	●	●
18	24 18	17 24 18	17 24 18	15 17 24 18

그림 8-19. 창문크기로 정의되는 프로쎄스의 작업모임

작업모임은 또한 시간의 함수이다. 만일 프로쎄스가 Δ시간단위이상 집행하고 한개의 페지만을 사용한다면 $|W(t,\Delta)|=1$로 된다. 작업모임은 또한 많은 서로다른 페지가 빨리 주소화된다면 그리고 창문크기를 허용한다면 프로쎄스의 페지수 N 만큼 크게 확대할수 있다. 그러므로

$$1 \le |W(t,\Delta)| \le \min(\Delta, N)$$

그림 8-20에서는 작업모임의 크기가 고정된 Δ의 값에 대해 시간을 변화시킬수 있다는것을 보여주고 있다. 많은 프로그람들에서 상대적으로 안정한 작업모임의 크기의 주기는 빠른 변화주기와 교체된다. 프로쎄스가 처음으로 집행을 시작할 때 새로운 페지를 참조함에 따라 점차적으로 작업모임을 구축한다. 결국 국소성의 원리에 의해 프로쎄스가 일정한 페지들의 모임에 기초하여 안정화된다. 그 후의 과도기간은 프로그람이 새로운 국소성에로 옮겨 간다는것을 반영한다. 이행단계기간에 과거의 국소성으로부터 일부 페지들이 창문 Δ안에 남아 있으면서 새로운 페지들이 참조됨에 따라 작업모임의 크기에서

큰 변동을 일으키게 된다. 창문에 페지참조들이 흘러갈 때 작업모임의 크기는 그것이 새로운 국소성으로부터 그 페지들만을 포함할 때까지 좁혀 진다.

작업모임에 대한 개념을 상주모임크기에 대한 전략을 끌어 내는데 사용할수 있다. 즉

1. 매개 프로쎄스에 대한 작업모임을 감시한다.
2. 프로쎄스의 상주모임으로부터 작업모임에 없는 페지들을 제거한다.
3. 프로쎄스는 그것의 작업모임이 주기억기에 있다면(즉 상주모임이 작업모임을 포함한다면) 집행할수 있다.

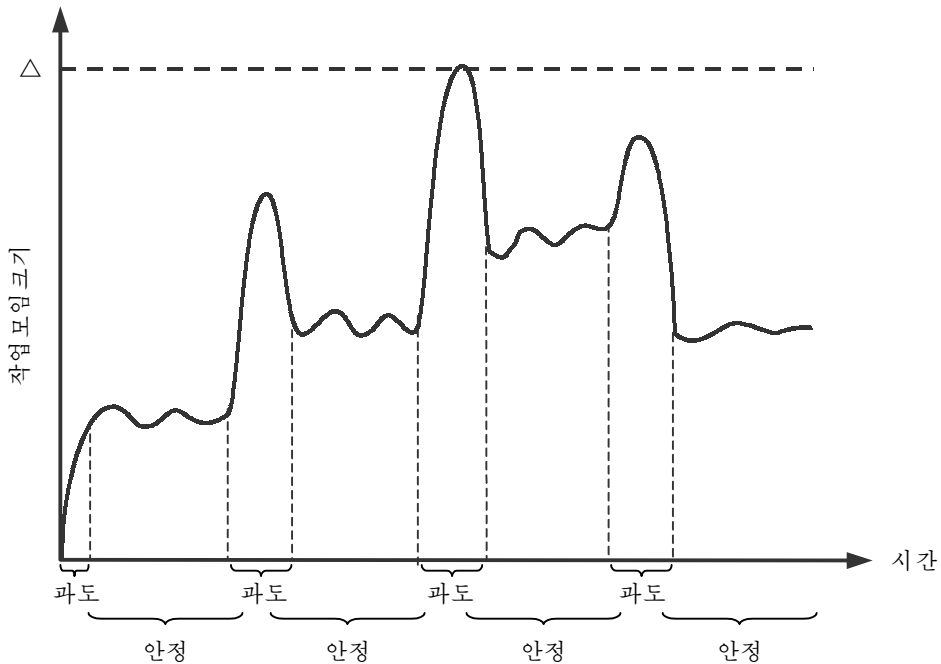



그림 8-20. 실지 작업모임의 크기에 대한 대표적그라프

이 전략이 인기를 끌고 있는데 그 리유는 그것이 국소성의 원리를 받아 들이고 페지부재를 최소화해 주는 기억기관리전략을 실현할수 있도록 해 주기때문이다. 작업모임전략에는 많은 문제점들이 있다. 즉

1. 과거는 언제나 미래를 예측하지 못한다. 작업모임의 크기와 성원은 모두 시간에 따라 변한다(그림 8-20 을 보시오.).
2. 매개 프로쎄스에서 작업모임을 실제적으로 측정하는것은 실천적인것이 못된다. 그 프로쎄스의 가상시간을 사용하는 매개 프로쎄스의 매개 페지참조시간을 찍어 줄 필요가 있으며 매개 프로쎄스에 대하여 페지들의 시간순서대기렬을 유지해야 한다.
3. Δ의 최적값은 알려 져 있지 않으며 경우에 따라 변한다.

그럼에도 불구하고 이 전략에 대한 갱신이 필요하며 많은 조작체계들이 작업모임전략을 받아 들이려고 한다. 이것을 하기 위한 하나의 방법은 정확한 페지참조들에가 아니

라 프로쎄스의 페지부재률에 중심을 두는것이다. 그림 8-11 ㄴ에서 설명하고 있는바와 같이 페지부재률은 프로쎄스의 상주모임크기를 증가시킴에 따라 떨어 진다. 작업모임의 크기는 그림에서 W 가 지적한 곡선우의 점에서 감소한다. 따라서 작업모임의 크기를 직접 감시하지 않고 페지부재률을 감시하여 비교할만한 결과를 얻을수 있다. 그 리유는 다음과 같다. 즉 만일 프로쎄스에서 페지부재률이 일정한 최소턱아래에 있다면 총적으로 체계는 그 프로쎄스에 해를 주지 않으면서(증가된 페지부재로 손해를 보지 않고) 프로쎄스에 더 작은 상주모임크기를 할당하여 리익을 얻을수 있다(더 많은 페지프레임들이 다른 프로쎄스에 대해 유용하기때문에). 만일 프로쎄스에서 페지부재률이 어떤 최대턱이상에 있으면 프로쎄스는 체계를 성능저하시키지 않으면서 증가된 상주모임크기로부터 리익을 얻을수 있다(페지부재들을 더 적게 일으킴으로써).

이 전략을 허용하는 알고리듬이 **페지부재빈도**(PFF)알고리듬이다[CHU72, GUPT78]. 알고리듬은 사용비트를 써서 기억기의 매개 페지와 련관된다. 그 비트는 페지가 접근을 받을 때 1 로 설정된다. 페지부재가 발생할 때 조작체계는 프로쎄스에서의 지난 페지부재로부터 가상시간을 표기하는데 페지참조에 대한 계수기를 사용하여 이것을 수행할수 있다. 어떤 턱 F 를 정의한다. 만일 지난 페지부재로부터의 시간이 F 보다 작다면 프로쎄스의 상주모임에 페지를 추가한다. 반면에 사용비트가 령인 모든 페지들을 버리고 그에 따라 상주모임을 축소시킨다. 같은 시간에 프로쎄스의 남아 있는 페지들에 대한 사용비트를 0 으로 재설정한다. 두개의 턱을 사용하여 전략을 개선할수 있다. 즉 상주모임을 증가시키는데 웃턱을 사용하며 상주모임크기를 축소시키는데 아래턱을 사용할수 있다.

페지부재사이의 시간은 페지부재률과 호상 련관되여 있다. 비록 페지부재률에 대한 실행평균을 유지하는것이 더 좋을듯 하여도 단일시간측정을 사용하는것이 페지부재률에 기초하여 상주모임의 크기를 결정할수 있는 합리적인 타협안으로 된다. 만일 페지완충법을 사용하여 그러한 전략을 실현한다면 결과적인 성능은 아주 좋아지게 된다.

그럼에도 불구하고 PFF방법에는 중요한 결함이 있는데 그것이 새로운 국소성에로의 옮김이 있는 과도기간동안 잘 수행되지 못한다는것이다. PFF에서 그것이 마지막으로 참조된 때로부터 F가상시간단위가 지나기전에 그 어떤 페지도 상주모임에서 떨어 져 나가지 않는다. 국소성사이의 이행기간에 페지부재들이 빨리 발생하는것은 과거의 국소성을 가지는 페지들을 제거하기전에 프로쎄스의 상주모임을 커지게 한다. 기억기요구의 급격한 첨두값들은 불필요한 프로쎄스의 비활성화 및 재활성화를 산생시킬수 있으며 그와 함께 대응하는 절환 및 교체에 걸리는 불필요한 간접소비시간을 산생시킬수 있다.

PFF방법의 간접소비시간과 류사한 상대적으로 낮은 간접소비시간을 가지는 국소성사이의 이행현상을 취급하는 방법이 바로 **가변간격표본작업모임**(VSWS))방책이다[FERR83]. VSWS방책은 지나간 가상시간에 기초하여 표본화하는 순간에 프로쎄스의 작업모임을 평가한다. 표본화간격초기에 프로쎄스에서의 모든 상주페지들의 사용비트들을 재설정한다. 마지막에 간격기간에 참조된 페지들만이 그것들의 사용비트를 설정한다. 이 페지들은 다음 간격 전기간 프로쎄스의 상주모임에 남아 있게 되며 다른것들은 제거된다. 매개 간격기간에 임의의 부재된 페지들은 상주모임에 추가되며 따라서 상주모임은 고정된채로 남아 있거나 그 간격기간에 증가된다.

VSWS 방책은 세개의 파라메터에 의해 구동된다. 즉

M: 표본화간격의 최소지속시간
L: 표본화간격의 최대지속시간

Q: 표본화순간들사이에서 발생이 허용되는 페지부재의 수

VSWS 방책은 다음과 같다. 즉

1. 마지막 표본화순간으로부터 가상시간이 L 에 이르면 프로쎄스를 중단시키고 사용비트들을 주사한다.
2. 만일 L 의 지나간 가상시간에 앞서 Q 개의 페지부재가 발생한다면
   ㄱ) 마지막 표본화순간으로부터 가상시간이 M 보다 작다면 지나간 가상시간이 M 에 이르러 프로쎄스를 중단시킬 때까지 기다리며 사용비트들을 주사한다.
   ㄴ) 마지막 표본화순간으로부터 가상시간이 M 보다 크거나 같다면 프로쎄스를 중단시키고 사용비트들을 주사한다.

파라메터값들은 표본화가 정상적으로 마지막 주사후에 Q 번째 페지부재가 발생할 때 시동되게끔 선택한다(경우 2 ㄴ). 다른 두개의 파라메터(M 와 L)는 례외적인 조건들에 대한 경계보호를 준다. VSWS 방책은 표본화빈도수를 증가시켜 급격한 국부성사이의 이행으로 인하여 생기는 첨두기억기요구들을 감소시키며 이로부터 페지화률이 증가할 때 쓰이지 않은 페지들이 상주모임에서 떨어 져 나가는 비률을 높인다. Bull 대형콤퓨터의 조작체계인 GCOS 8 에서 이 수법을 사용한 경험은 이 방법이 PFF 방법만큼 실현하기가 간단하면서 더 효과적이라는것을 지적하고 있다[PIZZ89].

지우기방책

지우기방책은 불러내기방책과 반대인데 변경된 페지를 2 차기억기에 언제 외부쓰기하겠는가를 판정하는 문제와 관련되여 있다. 두가지 일반적인 방도는 요구지우기와 미리지우기이다. **요구지우기**에서 페지는 치환용으로 선택되였을 때에만 2 차기억기에 외부쓰기된다. **미리지우기**방책은 변경된 페지들을 페지프레임이 요구되기전에 지우기하여 페지들을 일괄적으로 외부쓰기할수 있다.

어느 방책에서든 충분한 위험이 있다. 미리지우기에서 페지는 외부쓰기되지만 페지치환알고리듬이 그것을 제거하라고 지시할 때까지 주기억기에 남아 있는다. 미리지우기는 일괄적으로 페지들의 쓰기를 허용하지만 그것들의 대부분이 치환되기전에 다시 변경되였다는것을 발견하기 위해서만 수백 또는 수천개의 페지들을 외부쓰기한다는것은 의미가 거의 없다. 2 차기억기의 이송능력은 제한되여 있으며 불필요한 지우기조작으로 랑비되지 말아야 한다.

다른 한편 요구지우기에서 어지러운 페지의 쓰기는 선행하여 새로운 페지에서의 읽기와 결합한다. 이 수법은 페지의 쓰기를 최소화할수 있으나 페지부재를 당하는 프로쎄스가 비페색되기전에 두개의 페지이송을 기다려야 할수도 있다. 이것은 처리기의 사용률을 감소시킬수 있다.

더 좋은 방법은 페지완충법을 결합하는것이다. 이것은 다음의 방책을 받아들일수 있게 한다. 즉 치환가능한 페지들만을 지우기하는데 이때 지우기 및 치환조작들을 분리시킨다. 페지완충법에서 치환된 페지들을 두개의 목록 즉 변경 및 변경되지 않은 목록에 배치할수 있다. 변경된 목록에 있는 페지들은 주기적으로 일괄적으로 외부쓰기할수 있으며 변경되지 않은 목록에로 이동시킬수 있다. 변경되지 않은 목록에 있는 페지는 참조될 때 재생시켜 리용하든가 또는 프레임이 다른 페지에 할당될 때 지우기되든가 한다.

적재조종

적재조종은 주기억기에 상주하게 될 프로쎄스의 수를 결정하는 문제와 관련되여 있

는데 이것은 다중프로그람처리준위라고 불리왔다. 적재조종방책은 효과적인 기억기관리에서의 기준이다. 만일 임의의 어떤 시각에 너무 적은 프로쎄스들이 주기억기에 상주하고 있다면 모든 프로쎄스들이 페색될 기회가 많아 지며 많은 시간을 교체에 소비하게 된다. 다른 한편 너무 많은 프로쎄스들이 상주하고 있다면 평균적으로 매개 상주모임의 크기는 불충분하며 자주 부재가 일어 나게 된다. 결과 파도교체현상이 생긴다.

다중프로그람처리준위

파도교체현상을 그림 8-21 에서 설명하고 있다. 다중프로그람처리준위는 작은 값으로부터 증가함에 따라 처리기의 사용률이 증가하는데 그 리유는 모든 상주모임이 페색될 기회가 적기때문이다. 그러나 평균상주모임이 부족한 어떤 점에 이르게 된다. 이 점에서 페지부재의 수는 갑자기 늘어 나며 처리기의 사용률이 떨어 진다.

이 문제에 접근하기 위한 많은 방법들이 있다. 작업모임이나 페지부재빈도알고리듬은 암시적으로 적재조종과 통합된다. 단지 상주모임이 충분히 큰 프로쎄스들만이 집행을 허용받는다. 매개 능동프로쎄스에 요구되는 상주모임크기를 주는데서 방책은 자동적으로 그리고 동적으로 능동프로그람의 수를 결정한다.

Denning과 그의 동료들이 제기한 다른 방법[DENN80b]은 $L = S$기준으로 알려져있는데 이것은 다중프로그람처리의 준위를 조절하여 부재사이의 평균시간이 프로쎄스의 페지부재에 요구되는 평균시간과 같아 지도록 한다. 성능연구결과는 이것이 처리기의 사용률을 최대로 되게 하는 요점이라는것을 지적하고 있다. [LERO76]에서 제기한 류사한 효과를 가진 방책에서는 50%가 기준인데 이것은 대략 50%에서 페지화장치의 사용률을 유지한다. 또한 성능연구결과는 이것이 최대처리기사용률을 보장하는 기본문제라는것을 지적하고 있다.

또다른 방법은 이미 설명한 시계페지치환알고리듬을 받아 들이는것이다(그림 8-16). [CARR84]에서는 지시자가 프레임들의 순환완충기를 주사하는 속도를 감시하는것을 포함하는 전역범위를 사용하는 수법을 설명하고 있다. 만일 그 속도가 주어진 아래턱보다 낮다면 이것은 두 상태중의 하나 또는 모두를 지적한다. 즉

1. 지시자를 전진시킬데 대한 요청들을 발생시키는 페지부재들이 거의 일어나지 않는다.
2. 매개 요청에 대하여 지시자가 주사한 평균프레임의 수가 작아서 참조되고 있지 않는 많은 상주페지들이 있고 쉽게 치환할수 있다는것을 지적한다.

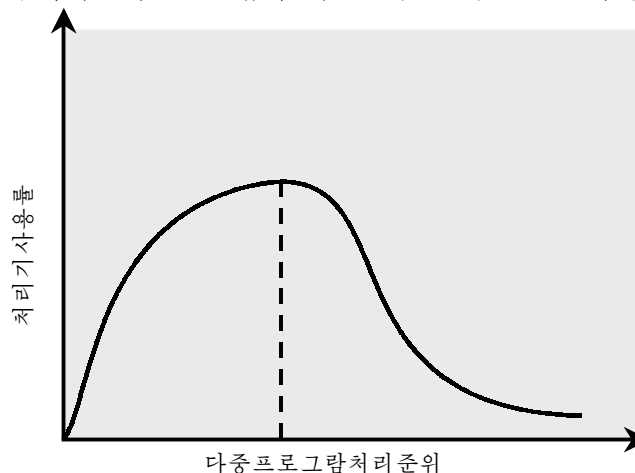



그림 8-21. 다중프로그람처리의 효과

두 경우에 모든 다중프로그람처리준위는 안전하게 증가시킬수 있다. 다른 한편 만일 지시자의 주사속도가 웃턱을 넘는다면 이것은 높은 부재률이든가 또는 치환할수 있는 페지들을 찾는데서 난관을 지적하는것으로서 다중프로그람처리준위가 너무 높다는것을 암시하는것으로 된다.

프로쎄스의 중단

만일 다중프로그람처리의 등급을 감소시킨다면 한개 또는 그이상의 현재 상주프로쎄스들을 중단시켜야 한다(교체하여 내보내야 한다.). [CARR84]에서는 여섯가지 가능성을 렬거하고 있다. 즉

- **최저우선권프로쎄스 :** 이것은 일정작성방책을 결정하는데 작용하며 성능문제와는 관련되지 않는다.
- **부재프로쎄스 :** 론거는 부재과제가 있는 자기의 작업모임을 상주시키게 하지 않을 가능성이 보다 크다는것이며 그것을 중단시키면 성능에 미치는 영향은 가장 작아 진다. 또한 이 선택은 어차피 페색시키려던 프로쎄스를 페색시키며 페지의 치환 및 입출력조작의 간접소비시간을 제거하므로 직접적인 리득을 준다.
- **최종활성프로쎄스 :** 이것은 자기의 작업모임을 상주시키는데 최소로 적합한 프로쎄스이다.
- **최소상주모임프로쎄스 :** 이것은 재적재하는데 가장 작은 노력을 요구한다. 그러나 그것은 작은 국소성을 가진 프로그람들에는 불리하다.
- **최대프로쎄스 :** 전면적으로 사용되는 기억기에서 가장 자유로운 페지를 얻어 곧 성공하지 못할 추가적인 해제를 한다.
- **최대나머지집행창문프로쎄스 :** 대부분의 프로쎄스일정작성방안들에서 프로쎄스는 새치기를 받아 준비대기렬의 끝에 배치되기전에 단지 일정한 시간동안 실행할수 있다. 이것은 최단처리시간우선일정작성규칙에 가깝다.

조작체계설계에 대한 많은 다른 분야에서와 같이 어느 방책을 설정하겠는가 하는 것은 판단에 관한 문제이며 집행되는 프로그람의 특성은 물론 조작체계에서 많은 다른 설계문제들에 관계된다.

제 3 절. UNIX와 Solaris의 기억기관리

UNIX는 독립적인 기계로 될것을 지향하고 있으므로 그의 기억기관리방안은 체계마다 변한다. UNIX의 초기판본은 아무런 단순한 가상기억기방안도 사용하지 않고 가변분할법을 사용하였다. SVR4 와 Solaris 2.x 를 포함하여 현재의 실현물들에서는 페지식가상기억기를 사용하고 있다.

SVR4 와 Solaris 에는 실제적으로 두개의 개별적인 기억기관리방안이 있다. 페지화체계는 주기억기에서 페지프레임들을 프로쎄스에 배정하고 또 페지프레임들을 디스크의 블로크완충기에 배정하는 가상기억기의 능력을 제공한다. 이것이 사용자프로쎄스와 디스크입출력에서 효과적인 기억기관리방안일지라도 페지식가상기억기방안은 핵심부에서의 기억기배정을 관리하는데는 적합하지 않다. 후자의 목적을 위해 **핵심부기억기배정**을 사용한다. 이 두가지 수법을 차례로 고찰한다.

페지화체계

자료구조

페지식가상기억기를 위해 UNIX는 최소의 조종으로 기계에 독립적인 많은 자료구조를 사용한다(그림 8-22와 표 8-5).

- **페지표 :** 대체로 프로쎄스당 하나의 페지표가 있는데 프로쎄스용으로 가상기억기에서 매개 페지에 하나의 입구점을 가진다.
- **디스크의 블로크서술자 :** 프로쎄스의 매개 페지와 련관된것은 가상페지의 디스크복사를 서술하는 표안의 입구점이다.
- **페지프레임의 자료표 :** 실제기억기의 매개 프레임을 서술하며 프레임번호에 의해 첨수화된다.
- **교체사용표 :** 매개 교체장치에 하나의 교체사용표가 있는데 장치의 매개 페지에 하나의 입구점을 가진다.

표 8-5에서 정의된 대부분의 마당은 자체해석적인것이다. 페지표입구점에서 나이마당은 프로그람이 프레임을 참조한 때로부터 얼마나 긴 시간이 흘렀는가를 가리킨다. 그러나 이 마당의 비트수와 갱신빈도수는 실현에 관계된다. 따라서 페지치환방책에서 이 마당에 대한 그 어떤 만능적인 UNIX사용도 없다.

디스크블로크서술자의 기억마당형태는 다음과 같은 리유로부터 요구된다. 즉 집행가능한 파일이 새로운 프로쎄스를 창조하는데 처음으로 사용될 때 파일에서의 프로그람과 자료의 일부분만 실제기억기에 적재되여 들어 갈수 있다. 후에 페지부재가 발생함에 따라 프로그람과 자료의 새로운 부분들이 적재된다. 가상기억기페지들이 교체에 사용될 장치들중의 하나에 있는 위치에 할당되는것은 다만 처음 적재될 때에만이다. 그때 조작체계는 프로그람이나 자료블로크를 처음으로 적재하기전에 페지프레임안의 위치들을 지우기할(0으로 설정할) 필요가 있는가 없는가 하는것을 알려 준다.

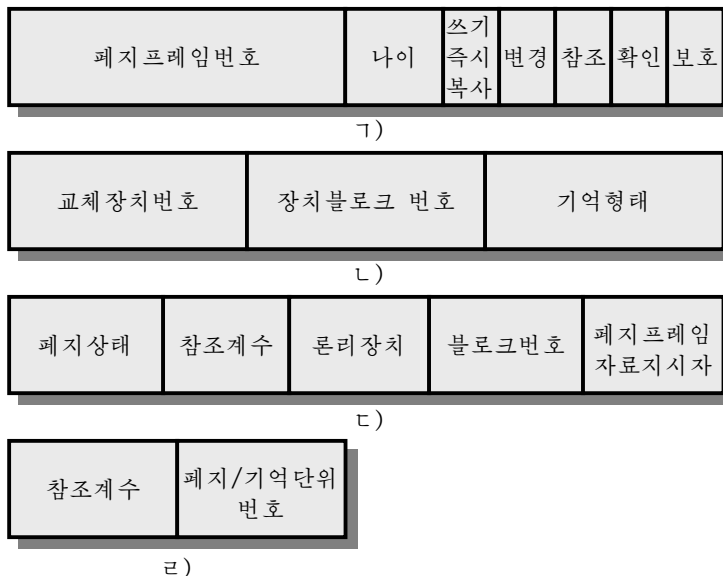



그림 8-22. UNIX SVR 4기억기관리의 양식: ㄱ-페지표입구점, ㄴ-디스크블로크서술자, ㄷ-페지프레임자료표입구점, ㄹ-교체사용표입구점

표 8-5. UNIX SVR4 기억기관리의 파라메터

페지표입구점

페지프레임번호
실제기억기에서 프레임을 가리킨다.

나이
페지가 참조되지 않고 기억기가 얼마나 오래동안 기억기에 있었는가를 가리킨다. 이 마당의 길이와 내용은 처리기에 관계된다.

쓰기즉시복사
한개이상의 프로쎄스가 페지를 공유할 때 설정한다. 만일 프로쎄스중의 한개를 페지에 써넣는다면 페지에 대한 개별복사는 페지를 공유하는 모든 다른 프로쎄스들에서 먼저 진행되여야 한다. 이것은 복사조작이 필요할 때까지 연기되게 하고 그것이 필요하지 않아 내보내는 경우들은 피하게 한다.

변경
페지가 변경되였다는것을 가리킨다.

참조
페지가 참조되였다는것을 가리킨다. 이 비트는 페지가 처음 적재될 때 0 으로 설정되고 페지치환알고리듬에 의해 주기적으로 재설정될수 있다.

확인
페지가 주기억기에 있다는것을 가리킨다.

보호
쓰기보호가 허용되는가 안되는가를 가리킨다.

디스크블로크서술자

교체장치번호
대응하는 페지를 유지하고 있는 2 차장치의 론리적장치번호. 이것은 한개이상의 장치를 교체하는데 사용할수 있다.

장치블로크번호
교체장치에서 페지의 블로크위치

기억형태
기억은 교체단위 또는 집행가능한 파일일수 있다. 후자의 경우에 배정될 가상기억기가 먼저 지우기되겠는가 말겠는가를 지시한다.

페지프레임자료표입구점

페지상태
프레임을 사용할수 있는가 또는 련관된 페지를 가지는가 하는것을 가리킨다. 후자의 경우에 페지의 상태가 교체장치집행가능한 파일 또는 진행중인 DMA 에서 명시된다.

참조계수
페지를 참조하는 프로쎄스들의 수

론리장치
페지의 복사를 담고 있는 론리장치

블로크번호
론리장치상에서 페지복사의 블로크위치

페지프레임자료지시자
자유페지의 목록과 페지하쉬대기렬상의 다른 페지프레임자료표입구점에 대한 지시자

교체사용표입구점

참조계수
교체장치상의 페지를 가리키는 페지표입구점들의 수

페지/기억단위번호
기억단위상의 페지식별자

페지치환

페지프레임자료표를 페지치환에서 사용한다. 몇가지 지시자들을 사용하여 이 표안에서 목록들을 창조한다. 사용할수 있는 모든 프레임들은 페지에 끌어 들이는데 사용할수 있는 자유로운 프레임들의 목록안에서 서로 련결된다. 사용할수 있는 페지의 수가 일정한 턱값아래로 떨어 질 때 핵심부는 보상하기 위해 많은 페지들을 끌어 들인다.

SVR 4에서 사용된 페지치환알고리듬은 시계방책알고리듬을 세련시킨것으로서(그림 8-16) 두방향시계알고리듬으로 알려 져 있다(그림 8-23). 알고리듬은 교체하여 내 보내기에 적합한(페쇄되지 않은) 기억기안의 매개 페지에 대한 페지표입구점안의 참조비트를 사용한다. 페지를 처음으로 끌어 들일 때 이 비트를 0으로 설정하며 페지가 읽거나 쓰기위해 참조될 때 1로 설정한다. 시계알고리듬에서 앞방향은 적합한 페지목록상에서 페지들을 훑어 보고 매개 페지상의 참조비트를 0으로 설정한다. 얼마후에 뒤방향은 같은 목록을 훑어 보고 참조비트를 검사한다. 그 비트가 1로 설정되여 있으면 그 페지가 앞방향 훑기가 있은후에 참조된것으로 되며 이 프레임은 무시한다. 만일 그 비트가 0으로 설정되여 있으면 앞방향과 뒤방향의 훑기사이의 시간간격에서 참조되지 않은것으로 되며 이 페지들을 페지화하여 내 보낼 목록에 배치한다.

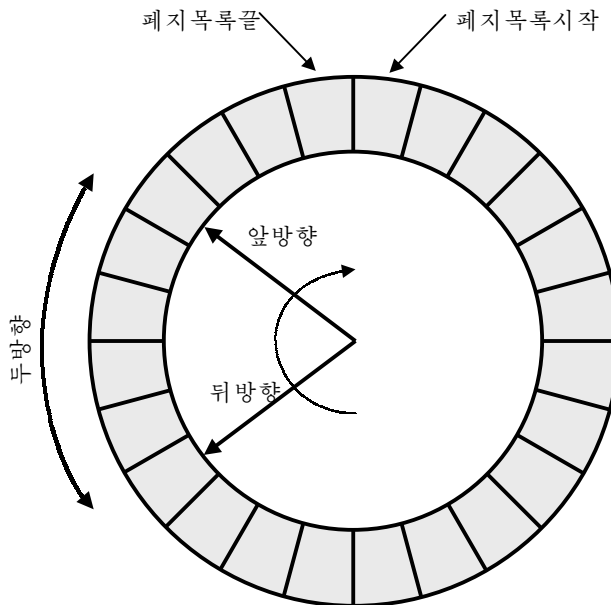



그림 8-23. 두방향시계페지치환알고리듬

두개의 파라메터가 알고리듬의 조작을 결정한다. 즉

- **주사속도 :** 초당 페지로서 두개의 방향으로 페지목록을 주사하는 속도
- **두방향 :** 앞방향과 뒤방향사이의 간격

이 두 파라메터들은 물리적기억기의 량에 기초하여 가동할 때 설정된 기정의 값들이다. 주사속도파라메터를 변경시켜 변화조건에 일치시킬수 있다. 파라메터는 자유기억기의 량이 최대자유값과 최소자유값들사이에서 변하는데 따라 느린 주사와 빠른 주사값(구성시간으로 설정한다.)사이에서 선형적으로 변한다. 다른 말로 자유기억기의 량이 축소

됨에 따라 시계방향은 더 많은 페지들을 자유롭게 하기 위해 더 빨리 움직인다. 두방향파라메터는 앞방향과 뒤방향사이의 간격을 결정하며 따라서 주사속도와 함께 페지가 적게 사용되므로 교체되여 나가기전에 어떤 페지를 사용할 기회의 창문을 결정한다.

핵심부기억기배정기

핵심부는 집행과정기간에 작은 표와 완충기들을 빈번히 생성하고 파괴하는데 그 매개가 동적기억기배정을 요구한다. [VAHA96]은 다음과 같은 실례들을 렬거하고 있다. 즉

- 경로이름변환루틴이 완충기를 배정하여 사용자공간으로부터 경로이름을 복사할수 있다.
- allocb()루틴이 독단적인 크기를 가지는 STREAMS 완충기들을 배정한다.
- 많은 UNIX 실현물들은 좀비구조를 배정하여 소멸된 프로쎄스들에 대하여 출구상태와 자원관리정보를 유지한다.
- SVR4 와 Solaris 에서 핵심부가 필요할 때 많은 객체들(proc 구조, vnode 들 및 파일서술자블로크와 같은)을 동적으로 배정한다.

이 블로크의 대부분은 대표적인 기계의 페지크기보다 대단히 작으며 따라서 페지화수법은 동적핵심부기억기배정에서 비효과적이다. SVR4 를 보면 제 7 장 제 2 절에서 설명한 동료체계를 변경하여 사용한다.

동료체계에서 기억기의 블로크를 배정하거나 해방하는 비용은 최적적합방책 및 최초적합방책의 비용에 비해 낮다[KNUT97]. 그러나 핵심부기억기관리인 경우에 배정과 해방조작이 가능한 빨리 이루어 져야 한다. 동료체계의 결함은 블로크들을 쪼각화하고 합치는데 요구되는 시간이다.

AT&T회사에서 Barkley와 Lee는 지연동료체계[BARK89]라고 하는 변종을 제의했는데 이것은 SVR4 를 위해 받아 들인 수법이다. 저자들은 UNIX가 흔히 핵심부방식에서 안정된 상태의 동작을 보여 준다는것을 관찰하였는데 이것은 곧 특정한 크기를 가진 블로크에 대한 요구량이 시간적으로 천천히 변한다는것이다. 따라서 크기가 2^i인 블로크를 해방하고 즉시에 크기가 2^{i+1} 인 블로크에로 합쳐 지면 핵심부가 다음에 크기가 2^i인 블로크를 요구할수 있는데 이것은 더 큰 블로크를 다시 쪼갤것을 요구할수 있다. 이 불필요한 결합과 쪼개기를 피하기 위해 지연동료체계는 그것이 요구할 때까지 결합을 미루고 그다음 가능한 많은 블로크들을 합친다.

지연동료체계는 다음과 같은 파라메터들을 사용한다. 즉

Ni = 크기가 2^i인 블로크의 현재의 수

Ai = 배정되는(차지되는) 크기가 2^i인 블로크의 현재의 수

Gi = 전역자유크기가 2^i인 블로크의 현재의 수이다. 이것들은 합치기에 적절한 블로크들이다. 만일 그러한 블로크의 협동이 전역적으로 자유롭게 된다면 두개의 블로크는 크기가 2^{i+1} 인 전역자유블로크에로 합쳐 진다. 표준적인 동료체계에서 모든 자유블로크들(구멍들)은 전역적으로 자유롭다고 간주할수 있다.

Li = 국부자유크기가 2^i인 블로크의 현재의 수이다. 이것은 합치기에 적절하지 않은 블로크들이다. 지어 그러한 블로크의 협동이 자유롭다고 하여도 두 블로크는 합쳐 지지 않는다. 오히려 국부자유블로크들이 그 크기의 블로크에 대한 앞으로의 요구를 예상하여 보류된다.

다음의 관계가 유지된다. 즉

$$Ni = Ai + Gi + Li$$

일반적으로 지연동료체계는 국부자유블로크들의 집결소를 유지하려고 하며 다만 국부자유블로크들의 수가 턱값을 넘으면 합치기를 개시한다. 국부자유블로크가 너무 많다면 요구를 만족시켜야 할 다음 준위에서 자유블로크들이 모자랄수 있는 기회가 생긴다. 블로크가 자유로워 질 때 대부분의 시간은 합치기를 발생하지 않으며 그래서 최소의 예산과 조작비용이 든다. 블로크가 배정될 때 국부적으로 및 전역적으로 자유로운 블로크들사이에서 명백한 차이가 이루어 지지 않으며 또 이것이 예산을 최소화한다.

합치기에 사용된 기준은 주어 진 크기의 국부자유블로크들이 그 크기의 배정된 블로크의 수를 넘지 말아야 한다는것이다(즉 $Li \le Ai$ 가 만족되여야 한다.). 이것은 국부자유블로크들의 확대를 제한하는 적당한 안내방향인데 [BARK89]에서의 실험은 이 방안이 현저한 절약을 가져 온다는것을 확증하고 있다.

이 방안을 실현하기 위해 다음과 같이 지연변수를 정의한다. 즉

$$Di = Ai - Li = Ni - 2Li - Gi$$

그림 8-24에서는 그 알고리듬을 보여 주고 있다.

Di의 초기값은 0이다.
조작후에 Di의 값은 다음과 같이 변경된다.

(I) 만일 다음조작이 블로크배정요청이면:
만일 자유블로크가 있으면 배정하기 위해 한개를 설정한다.
만일 선택된 블로크가 국부적으로 자유로우면
그러면 D*i*:=Di+2
아니면 D*i*:=Di+1
한편
우선 보다 큰 한개를 두개로 쪼개여 두개의 블로크를 만든다(재귀조작).
한개를 배정하고 다른 국부적으로 자유로운것에 표식을 단다.
D_i는 변화시키지 않은채로 남겨 둔다(그러나 D_i는 재귀호출로 하여 다른 블로크크기로 변화시킬 수 있다.).

(II) 만일 다음 조작이 블로크자유요청이면
D*i*≧2인 경우
그것에 국부자유표식을 하고 그것을 국부적으로 자유롭게 한다.
D*i*:=Di-2
D*i*=1인 경우
그것에 전역자유표식을 하고 그것을 전역으로 자유롭게 한다; 가능하면 합친다.
Di:=0
D*i*=0인 경우
그것에 전역자유표식을 하고 그것을 전역으로 자유롭게 한다; 가능하면 합친다.
크기가 2^i인 국부적으로 자유로운 한개의 블로크를 선택하고 그것을 전역으로 자유롭게 한다; 가능하면 합친다.
Di:=0

그림 8-24. 지연동료체계의 알고리듬

제 4 절. LINUX의 기억기관리

LINUX는 다른 UNIX실현물들의 기억기관리방안중에서 많은 기능들을 공유하고 있으면서도 자기자체의 통일적인 특징을 가지고 있다. 총적으로 LINUX의 기억기관리방안

은 아주 복잡하다[DUBE98]. 여기서는 간단한 개괄을 준다.

LINUX가상기억기

가상기억기의 주소화

LINUX는 세개 준위의 페지표구조를 사용하는데 그것은 다음의 형태로 된 표들로 이루어 져 있다(매개 개별적인 표의 크기는 한페지이다.). 즉

- **페지등록부 :** 능동프로쎄스는 단일한 페지등록부를 가지는데 한페지크기이다. 페지등록부의 매개 입구점은 페지중간등록부들중에서 한페지를 지적한다. 페지등록부는 능동프로쎄스용으로 주기억기에 있어야 한다.
- **페지중간등록부 :** 페지중간등록부는 다중페지들을 관리할수 있다. 페지중간등록부의 매개 입구점은 페지표의 한페지를 지적한다.
- **페지표 :** 페지표는 또한 다중페지들을 관리할수 있다. 매개 페지표입구점은 프로쎄스의 한개의 가상페지에 귀착된다.

이 세개의 준위페지표구조를 사용하기 위해 LINUX 에서는 가상주소를 네개의 마당으로 구성되여 있는것으로 본다. 제일 왼쪽(가장 중요한) 마당을 페지등록부에 들어 가는 첨수로 사용한다. 다음마당은 페지중간등록부에 들어 가는 첨수로 봉사한다. 세번째 마당은 페지표에 들어 가는 첨수로 동작한다. 네번째 마당은 선택된 기억기의 페지안에서의 편위를 준다.

LINUX 의 페지표구조는 가동환경에 무관계하게 64bit Alpha 처리기를 수용하도록 설계되였는데 이것은 세개의 페지화준위를 하드웨어적으로 지원한다. 64bit 주소를 가지고 Alpha 상에서 단지 두개의 페지준위를 사용하는것은 매우 큰 페지표와 등록부를 만들게 한다. 32bit 펜티움/x86 구성방식은 2 준위하드웨어페지화기구를 가지고 있다. LINUX 쏘프트웨어는 페지중간등록부의 크기를 하나로 정의하여 2 준위방안을 수용한다.

페지배정

기억기에 대해 페지를 읽어 들이거나 써내기하는데서 효과성을 높이기 위해 LINUX 는 페지프레임의 련속된 블로크들에 사영된 련속된 페지블로크를 취급하는 수법을 정의한다. 이 목적을 위해서 동료체계를 사용한다. 핵심부는 고정된 크기를 가진 련속된 페지프레임그룹의 목록을 유지하고 있는데 그룹은 1, 2, 4, 8, 16 또는 32 개의 페지프레임으로 구성할수 있다. 페지가 주기억기에 배정되고 삭제됨에 따라 사용할수 있는 그룹은 동료알고리듬을 사용하여 분할되거나 합동된다.

페지치환알고리듬

LINUX 의 페지치환알고리듬은 제 8 장 제 2 절에서 설명한 시계알고리듬에 기초하고 있다(그림 8-16 을 보시오). 단순한 시계알고리듬에서 사용비트와 변경비트는 주기억기안의 매개 페지와 련관되여 있다. LINUX 방안에서 사용비트는 8 비트의 나이변수와 교체된다. 페지가 접근을 받을 때마다 나이변수가 증가한다. 내적으로 LINUX 는 전역페지집결소를 주기적으로 쭉 훑어 보며 그것이 주기억기에서 모든 페지들을 쭉 순환함에 따라 매개 페지에서의 나이변수를 감소시킨다. 나이가 《0》인 페지는 《늙은》 페지로서 그것은 일정한 시간안에 참조되지 않았으며 가장 좋은 치환후보로 된다. 나이의 값이

커지면 커질수록 최근시간에 더 자주 페지를 사용하며 치환용으로는 점점 더 적합하지 않다. 그러므로 LINUX 알고리듬은 최소빈도사용방책의 형태이다.

핵심부기억기배정

LINUX 에서 핵심부기억기배정의 기본은 기억기배정수법으로서 가상기억기관리에 쓰인다. 가상기억기방안에서와 같이 동료알고리듬을 사용하여 핵심부에서의 기억기를 한개 또는 그이상의 페지단위들로 배정하고 해방할수 있게 하되 이런 방식으로 배정할수 있는 최소의 기억기량은 한페지이기때문에 페지배정에 홀로는 효과적이지 못하다. 그것은 핵심부가 작은 단기기억기덩어리들을 기수크기들로 요구하기때문이다. 이 작은 덩어리들을 수용하기 위해 LINUX 는 배정된 페지안에서 기억판배정[BONW94]이라는 수법을 사용한다. 펜티움/x86 기계상에서 페지크기는 4kbyte 이며 페지안에서 덩어리를 32, 64, 128, 252, 508, 2040 및 4080byte 의 크기로 배정할수 있다.

기억판배정기는 상대적으로 복잡하며 여기서는 구체적으로 고찰하지 않는데 [VAHA96]에서 상세히 설명하고 있다. 본질적으로 LINUX 는 덩어리의 매개 크기에 대해 하나씩 련결목록모임을 유지하고 있다. 동료알고리듬과 류사한 방식으로 덩어리를 쪼개고 합칠수 있으며 목록들사이에서 적당히 이동시킬수 있다.

제 5 절. Windows 2000의 기억기관리

Windows 2000(W2K)의 가상기억기관리자는 기억기를 배정하는 방법과 페지화수행방법을 조종한다. 기억기관리자는 다양한 가동환경들에서 동작하며 4kbyte∼64kbyte 범위의 페지크기를 사용하도록 설계되여 있다. Intel, PowerPC 및 MIPS 가동환경들은 페지당 4096byte 를 가지고 있으며 DEC Alpha 의 가동환경들은 페지당 8192byte 를 가지고 있다.

W2K 가상주소사영

매개 W2K 의 사용자프로쎄스는 개별적인 32 비트주소공간을 보는데 프로쎄스당 4Gbyte 의 기억기를 허용한다. 이 기억기의 일부분은 조작체계용으로 암시적으로 예약되여 있어 매개 사용자는 실제적으로 2Gbyte 의 사용할수 있는 가상주소공간을 가지며 모든 프로쎄스는 같은 2Gbyte 의 체계공간을 유지한다. 사용자공간을 3Gbyte 로 증가시키고 체계용으로 1Gbyte 를 남겨 놓게 할수 있는 선택방안이 있다. W2K 의 문서는 이 특징이 수기가비트의 RAM 을 가진 봉사기상에서 큰 기억기의 철저한 응용프로그람을 지원할것을 의도하고 있다는것과 보다 큰 주소공간을 사용하면 결정지원 또는 자료채취와 같은 응용프로그람의 성능을 훨씬 개선할수 있다는것을 지적하고 있다.

그림 8-25 에서는 사용자프로쎄스가 본 암시적인 가상주소공간을 보여 주고 있다. 그것은 네개의 구역으로 구성되여 있다.

- Ox00000000∼Ox0000FFFF : 프로그람작성자들이 NULL 지시자를 할당하는것을 방조하기 위해 예비로 내 놓는다.
- Ox00010000∼Ox7FFFFFFF : 사용할수 있는 사용자주소공간. 이 공간을 주기억기에 적재시킬수 있는 페지들로 나눈다.

• Ox7FFF0000～Ox7FFFFFFF : 사용자가 접근할수 없는 감시페지. 이 페지는 조작체계가 경계바깥에 대한 지시자의 참조를 검사하기 쉽게 해 준다.
• Ox80000000～OxFFFFFFFF : 체계주소공간. 2Gbyte 의 프로쎄스는 W2K 의 집행부, 마이크로핵심부 및 장치구동기용으로 사용한다.

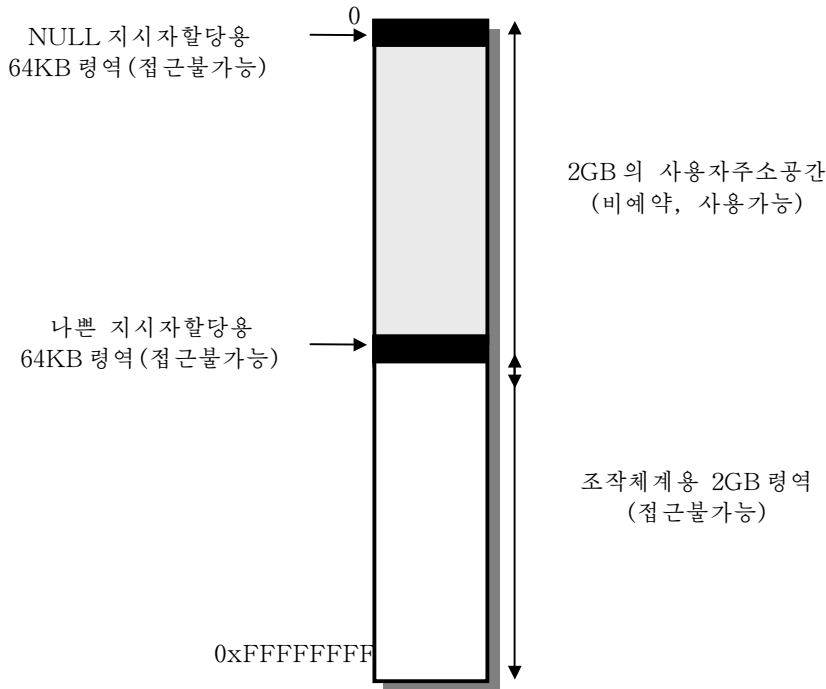



그림 8-25. Windows 2000의 암시적가상주소공간

W2K 페지화

프로쎄스를 창조할 때 원리적으로는 2Gbyte(-128kbyte)의 전체 사용자공간을 사용할수 있다. 이 공간을 고정크기의 페지들로 나누고 그중 임의의것을 주기억기에 끌어다 넣을수 있다. 실천적으로 계산을 간단히 하기 위해 페지는 세개 상태중의 하나에 있을수 있다. 즉

• **사용가능상태 :** 프로쎄스가 현재 사용하지 않는 페지들
• **예약상태 :** 가상기억기관리자가 프로쎄스용으로 설정해 놓지만 사용될 때까지 프로쎄스의 기억기할당몫을 계산에 넣지 않는 련속된 페지들의 모임. 프로쎄스가 기억기에 쓰려고 할 때 예약된 기억기의 일부를 프로쎄스에 허용해 준다.
• **허용상태 :** 가상기억기관리자가 페지화파일에 공간을 내 놓은 페지들(실례로 그것들을 주기억기에서 제거할 때 페지들에 써 넣는 디스크파일)

예약 및 허용된 기억기사이를 구별해 주는것이 좋은데 그 리유는 (1) 특정한 프로쎄스용으로 내 놓은 디스크공간의 량을 최소화하고 그 공간이 다른 프로쎄스에 대해서는 자유롭게 해 주기때문이며 (2) 스레드나 프로쎄스가 필요에 따라 빨리 배정할수 있는 기억기의 량을 선언할수 있게 해주기때문이다.

W2K 가 사용하는 상주모임관리방안은 가변배정, 국부범위이다(표 8-4 를 보시오.).

프로쎄스가 처음으로 활성화 될 때 그것은 주기억기의 일정한 수의 페지프레임들에 작업모임으로 할당된다. 프로쎄스가 기억기에 없는 페지를 참조할 때 프로쎄스의 상주페지중의 하나를 교체하여 내 보내고 새로운 페지를 끌어들인다. 능동프로쎄스의 작업모임은 다음과 같은 일반적인 규정들을 사용하여 조절된다. 즉

- 주기억기가 충분할 때 가상기억기관리자는 능동프로쎄스의 상주모임을 커지게 한다. 이를 위해 페지부재가 발생할 때 새로운 페지를 주기억기에 끌어 들이지만 그 어떤 보다 낡은 페지를 교체하여 내 보내지 않음으로써 프로쎄스의 상주모임을 한페지만큼 증가시킨다.
- 기억기가 부족해 질 때 가상기억기관리자는 능동프로쎄스의 작업모임가운데서 최근에 보다 적게 사용된 페지들을 이동시켜 체계에서 기억기를 회복시키고 상주모임의 크기를 감소시킨다.

요약, 기본용어 및 복습문제

처리기 및 입출력기능을 효과적으로 사용하자면 주기억기에 가능한 많은 프로쎄스들을 보존하여야 한다. 또한 프로그람작성자들을 프로그람개발에서 크기제한으로부터 해방시켜 주어야 한다.

이 두가지 조건을 모두 만족해 주는 방도는 바로 가상기억기이다. 가상기억기를 사용하면 모든 참조주소가 론리적참조로 되는데 그것은 실행시에 실제주소로 변환된다. 이것은 프로쎄스를 주기억기의 임의의 곳에 배치할수 있으며 그 위치가 시간에 따라 변할수 있다. 가상기억기는 또한 프로쎄스를 쪼각들로 분할할수 있다. 이 쪼각들은 집행기간에 주기억기에 련속되여 있을것을 요구하지 않으며 실지로 프로쎄스의 모든 쪼각들이 집행기간에 주기억기에 있을 필요도 없다.

가상기억기를 제공하는 두가지 기본 방법은 페지화와 토막화이다. 페지화에서 매개 프로쎄스는 상대적으로 작고 고정된 크기를 가진 페지들로 분할된다. 토막화는 크기가 변하는 쪼각들을 사용할수 있게 해 준다. 또한 단일한 기억기관리방안으로 토막화와 페지화를 조합할수 있다.

가상기억기관리방안은 하드웨어와 쏘프트웨어지원을 모두 요구한다. 하드웨어지원은 처리기가 보장한다. 그 지원은 가상주소를 물리적주소에로 동적변환하는것 및 참조된 페지나 토막이 주기억기에 없을 때 새치기를 발생시키는것을 포함한다. 그러한 새치기는 조작체계에서 기억기관리쏘프트웨어를 기동시킨다.

많은 설계문제들이 기억기관리를 위한 조작체계의 지원과 련관되여 있다. 즉

- **불러내기방책 :** 프로쎄스의 페지를 요구에 따라 끌어 들일수 있으며 또는 미리페지화방책을 사용할수 있는데 그것은 한번에 많은 페지를 끌어 들이는것으로 하여 입력동작을 클라스터화한다.
- **배치방책 :** 순수한 토막화체계에서 끌어 들이는 토막은 기억기에서 사용할수 있는 공간에 맞아야 한다.
- **치환방책 :** 기억기가 찼을 때 어느 페지 또는 페지들을 치환하겠는가를 결정해야 한다.
- **상주모임관리 :** 조작체계는 프로쎄스가 교체되여 들어 올 때 특정한 프로쎄스에 배정할 기억기의 량을 결정해야 한다. 이것은 프로쎄스를 창조할 때 정적으로 배정할수 있거나 동적으로 변화시킬수 있다.

- **지우기방책 :** 변경된 프로쎄스의 페지들은 치환할 때 외부쓰기할수 있거나 치환방책을 사용할수 있는데 이것은 한번에 많은 페지를 치환하는것으로 하여 출력동작을 클라스터화한다.
- **적재조종 :** 적재조종은 임의의 주어 진 시간에 주기억기에 상주하게 될 프로쎄스의 수를 결정하는것과 관련되여 있다.

기본용어

련상사영	페지	상주모임관리
요구페지화	페지부재	토막
외부쪼각	페지배치방책	토막표
불러내기방책	페지치환방책	토막화
프레임	페지표	기억판배정
하쉬표	페지화	과도교체
하쉬법	미리페지화	변환미리보기완충기
내부쪼각	실제주소	가상기억기
국소성	상주모임	작업모임

복습문제

1. 단순페지화와 가상기억기페지화의 차이는 무엇인가?
2. 하쉬법을 설명하시오.
3. 가상기억기를 사용하는데서 국소성의 원리가 왜 결정적인가?
4. 페지표입구점에서 대체로 보게 되는것은 어떤 요소들인가? 매개 요소를 간단히 정의하시오.
5. 변환미리보기완충기의 목적은 무엇인가?
6. 선택할수 있는 페지불러내기방책을 간단히 정의하시오.
7. 상주모임관리와 페지치환방책사이의 차이는 무엇인가?
8. FIFO 방책과 시계페지치환알고리듬사이의 관계는 무엇인가?
9. 페지완충법에 의하여 무엇이 수행되는가?
10. 전역치환방책과 고정배정방책을 왜 조합시킬수 없는가?
11. 상주모임과 작업모임사이의 차이는 무엇인가?
12. 요구지우기화와 미리지우기화의 차이는 무엇인가?

참 고 문 헌

예상한바와 같이 가상기억기는 조작체계에 관한 대부분의 책들에서 좋은 호평을 받고 있다. [MILE92]에서는 여러가지 연구분야들에 대한 좋은 개요를 주고 있다. [CARR84]에서는 성능문제에 대하여 훌륭하고 깊이 있게 고찰하고 있다. 고전적론문인 [DENN70]은 여전히 읽을 가치가 있다. [DOWD93]에서는 여러가지 페지치환알고리듬에 대한 교훈적인 성능분석을 주고 있다. [JACO98a]에서는 가상기억기설계에서의 문제점들을 잘 개괄하고 있다. [JACO98b]에서는 마이크로처리기들에서 가상기억기의 하드웨어구성문제를 서술하고 있다.

[IBM86]에서는 MVS의 가상기억기방책을 최적화하는데서 싸이트관리자가 사용할수 있는 도구들과 선택방식들을 구체적으로 고찰하고 있다. 론문에서는 이 문제의 복잡성을 설명하고 있다.

[VAHA96]은 여러가지 UNIX의 변종들에서 사용된 기억기관리방안을 가장 훌륭하게 취급한것중의 하나이다.

CARR84 Carr, R. Virtual Memory Management. Ann Arbor, MI: UMI Research Press, 1984.

DENN70 Denning, P. "Virtual Memory." Computing Surveys, September 1970.

DOWD93 Dowdy, L., and Lowery, C. P.S. to Operating Systems. Upper Saddle River, NJ: Prentice Hall, 1993.

IBM86 IBM National Technical Support, Large Systems. Multiple Virtual Storage (MVS) Virtual Storage Tuning Cookbook. Dallas Systems Center Technical Bulletin G320-0597, June 1986.

JAC098a Jacob, B., and Mudge, T. "Virtual Memory: Issues of Implementation." Computer, June 1998.

JAC098b Jacob, B., and Mudge, T. "Virtual Memory in Contemporary Microprocessors." IEEE Micro, August 1998.

MILE92 Milenkovic, M. Operating Systems: Concepts and Design. New York: McGraw-Hill, 1992.

VAHA96 Vahalia, U. UNIX Internals: The New Frontiers. Upper Saddle River, NJ: Prentice Hall, 1996.

련 습 문 제

1. 현재 처리기상에서 집행중인 프로쎄스에 대한 페지표가 다음의것과 같다고 가정하자. 모든 수는 10진수이며 모든것은 0으로부터 시작하여 수자를 매기였고 모든 주소는 기억기의 바이트주소이다. 페지크기는 1024byte이다.

가상페지수	유효비트	참조비트	변경비트	페지프레임수
0	1	1	0	4
1	1	1	1	7
2	0	0	0	-
3	1	0	0	2
4	0	0	0	-
5	1	0	1	0

ㄱ) 일반적으로 CPU에서 발생한 가상주소가 물리적인 주기억기의 주소로 어떻게 변환되는지 정확히 설명하시오.

ㄴ) 다음과 같은 매개 가상주소는 어떤 물리주소에 대응하는가?(페지부재가 있다고 해도 처리하지 마시오.)

(1) 1052
(2) 2221
(3) 5499

2. 프로쎄스가 그것에 배정된 4 개의 프레임을 가지고 있다(다음의 수들은 모두 10 진수이며 모든것은 0 으로부터 시작하여 수를 매기였다.). 페지를 매개 페지프레임에 마지막으로 적재하는 시간, 매개 페지프레임의 페지에 대한 마지막 접근시간, 매개 페지프레임의 가상페지번호, 매개 페지프레임에 대한 참조(R) 및 변경(M)비트들은 제시된것과 같다(시간들은 시각 0 에서 프로쎄스의 시작으로부터 사건까지의 박자수이다. 그 사건으로부터 현재까지의 박자수는 아니다.).

가상페지번호	페지프레임	적재된 시간	참조된 시간	R 비트	M 비트
2	0	60	161	0	1
1	1	130	160	0	0
0	2	26	162	1	0
3	3	20	163	1	1

가상페지 4 에서 페지부재가 발생하였다. 아래의 기억기관리방책에 대하여 어느 페지프레임이 치환된 내용을 가지게 되는가? 매 경우에 그 리유를 설명하시오.

ㄱ) FIFO(선입선출)
ㄴ) LRU(최대미사용)
ㄷ) 시계
ㄹ) 최적(다음의 참조렬을 사용하시오.)
ㅁ) 페지부재직전에 기억기에 대해 앞에서 언급한 상태가 주어 진 조건에서 다음의 가상페지참조렬을 고찰하시오. 즉

4, 0, 0, 0, 2, 4, 2, 1, 0, 3, 2

고정배정방책대신에 창문크기가 4 인 작업모임방책을 사용한다면 얼마나 많은 페지부재가 발생하겠는가? 언제 매개 페지부재가 발생하게 되는지 명백히 설명하시오.

3. 프로쎄스가 다섯개의 페지 A, B, C, D, E 를 다음과 같은 순서로 참조한다.

A; B; C; D; A; B; E; A; B; C; D; E

치환알고리듬은 선입선출법이라고 가정하고 세개의 페지프레임을 가지는 빈 주기억기로 시작하는 참조의 순서기간에 페지의 이송번호를 찾으시오. 4 개의 페지프레임에 대하여 반복하시오.

4. 프로쎄스가 디스크상에 있는 여덟개의 가상페지를 포함하고 있으며 주기억기에서 네개의 페지프레임에 대한 고정배정을 할당받는다. 다음의 페지추적이 발생한다. 즉

1,0,2,2,1,7,6,7,0,1,2,0,3,0,4,5,1,5,2,4,5,6,7,6,7,2,4,2,7,3,3,2,3

ㄱ) LRU 치환방책을 사용하여 네개의 페지프레임에 상주하는 성공적인 페지 들을 찾아 보시오. 주기억기에서 명중률을 계산하시오. 프레임들은 초기에 비여 있다고 가정하시오.
ㄴ) FIFO 치환방책에 대하여 ㄱ를 반복하시오.

ㄷ) 두개의 명중률과 특정한 추적에 대하여 LRU 에 근사한 FIFO 사용의 효과성을 비교하시오.

5. VAX에서 사용자페지표들이 체계공간의 가상주소에 배치되여 있다. 주기억기보다 가상기억기에서 사용자페지표를 가지고 있는것의 우점은 무엇인가? 결함은 무엇인가?
6. 프로그람명령문

```
for ( i = 1; i <= n; i ++ )
        a[i] = b[i] + c[i];
```

이 페지크기가 1000 단어인 기억기에서 집행된다고 하자. n = 1000 이라고 하자. 충분한 범위의 등록기 대 등록기명령을 가지고 있으며 첨수등록기를 가지는 기계를 사용하여 앞의 명령문을 실현하기 위한 가상적인 프로그람을 작성하시오.
7. IBM System/370 구성방식은 2 준위기억기구조를 사용하며 토막화방법은 이 장의 앞부분에서 설명한 많은 특징들이 부족하기는 하지만 토막과 페지들로 두개의 준위를 이루고 있다. 기본 370 구성방식에서 페지크기는 2kbyte 나 4kbyte 중의 어느 하나로 될수 있으며 토막의 크기는 64kbyte 나 1Mbyte 중의 어느 하나에 고정되여 있다. 370/XA 와 370/ESA 구성방식에서 페지크기는 4kbyte 이며 토막의 크기는 1Mbyte 이다. 이 방안을 토막화의 어떤 우점들이 빠졌는가? 370 에서 토막화의 리익은 무엇인가?
8. 페지크기를 4kbyte 로 그리고 페지표입구점이 4kbyte 를 가진다고 가정하고 만일 웃준위페지표가 단일페지에 적합하다면 64 비트주소공간을 사영하는데 몇개의 페지표가 필요한가?
9. 페지에 기초하여 수행되는 기억기사영을 가지고 있으며 단일준위의 페지표를 사용하는 어떤 체계를 고찰하자. 필요한 페지표는 항상 기억기에 있다고 가정하자.

ㄱ) 기억기참조가 200ns 걸린다면 페지식기억기참조는 얼마나 오래 걸리는가?
ㄴ) 이제 명중과 실패에 관하여 20ns 의 간접소비시간을 부과하는 MMU 를 추가하자. 만일 MMU TLB 에서 모든 기억기참조의 85%가 명중한다고 가정하면 유효기억기접근시간(EMAT)은 얼마인가?
ㄷ) TLB 명중률이 EMAT 에 어떻게 영향을 주는가에 대하여 설명하시오.

10. 초기에 모두 비여 있는 M 개의 프레임들로 된 작업모임을 가진 프로쎄스에서의 페지참조렬을 고찰하자. 페지참조렬은 그 안에서 명백한 페지번호들을 가진 길이가 P 인 렬이다. 임의의 페지치환알고리듬에 대하여

ㄱ) 페지부재수의 아래한계는 얼마인가?
ㄴ) 페지부재수의 웃한계는 얼마인가?

11. 페지치환알고리듬설명에서 한 저자는 원궤도를 도는 제설장치와 상사적인것을 설명하고 있다. 눈은 고르롭게 궤도에 내리고 있으며 외토리제설장치는 일정한 속도로 련속 궤도를 돈다. 궤도에서 제설된 눈은 체계에서 없어 진다.

ㄱ) 제 8 장 제 2 절에서 설명한 페지치환알고리듬중에서 어느것이 쓸모 있는 상사적인것으로 되는가?
ㄴ) 질문에서 페지치환알고리듬의 동작에 대한 이 상사가 무엇을 암시하는가?

12. S/370 구성방식에서 기억기의 열쇠는 실제기억기의 매개 페지크기의 프레임과 관련되여 있는 조종마당이다. 페지치환와 관련된 열쇠의 두개 비트는 참조비트와 변화비트이다. 참조비트는 프레임안에서 임의의 주소가 읽거나 쓰기하여 접근을 받을 때 1 로 설정되며 새로운 페지가 프레임에 적재될 때 0 으로 설정된다. 변화비트는 쓰기조작이 프레임안의 임의의 위치에 관하여 수행될 때 1 로 설정된다. 참조비트만을 사용하여 어느 페지프레임이 가장 최근에 사용되는가를 결정하는 방법을 제기하시오.
13. 상주모임관리방책의 성능에서 기본은 값 Q이다. 경험은 프로쎄스에서 Q의 값을 고정시키면 각이한 집행상태들의 페지부재빈도수에서 현저한 차이가 생긴다는것을 보여 주고 있다. 더우기 서로다른 프로쎄스에 단일한 값 Q를 사용하면 극적으로 차이나는 페지부재빈도률이 발생한다. 이 차이들은 프로쎄스의 수명주기동안Q의 값을 동적으로 조절하는 기구가 알고리듬의 동작을 개선할수 있다는것을 강하게 시사하고 있다.이 목적을 실현하기 위한 단순한 기구를 제안하시오.
14. 어떤 과제를 네개의 같은 크기를 가진 토막들로 분할한다는것과 체계가 매개 토막용으로 입구점이 여덟개인 페지서술자표를 만든다고 가정하자. 따라서 체계는 토막화와 페지화의 조합을 가진다. 또한 페지크기는 2kbyte 라고 가정하자.

   ㄱ) 매개 토막의 최대크기는 얼마인가?

   ㄴ) 과제에서의 최대론리주소공간은 얼마인가?

   ㄷ) 물리적위치 00021ABC 의 요소는 이 과제에 의하여 접근을 받는다. 과제가 그것을 위해 산생시키는 론리적주소의 양식은 무엇인가? 체계의 최대물리적 주소공간은 얼마인가?

15. 1Mbyte 의 물리적주소공간에 사용된 페지화된 론리주소공간(2kbyte 씩 32 개의 페지로 구성된)을 고찰하자.

   ㄱ) 처리기의 론리적주소에 대한 양식은 무엇인가?
   ㄴ) 페지표의 길이와 너비는 얼마인가?(《접근권한》비트는 고려하지 않는다.)
   ㄷ) 물리적기억공간을 절반으로 줄인다면 페지표에 미치는 영향은 무엇인가?

16. 콤퓨터가 캐쉬, 주기억기 및 가상기억기용으로 쓰이는 디스크를 가지고 있다. 참조된 단어가 캐쉬안에 있다면 그에 접근하는데 20ns 가 필요하다. 그것이 캐쉬가 아니라 주기억기에 있다면 그것을 캐쉬에 적재하는데 60ns 가 필요하며 그다음에 참조를 다시 시작한다. 단어가 주기억기에 없다면 디스크에서 단어를 불러 내는데 12ms 가 걸리며 이어 그것을 캐쉬에 복사하는데 60ns 걸리며 그다음에 참조를 다시 시작한다. 캐쉬명중률은 0.9 이고 주기억기의 명중률은 0.6 이다. 이 체계에서 참조된 단어에 접근하는데 요구되는 평균시간은 ns 로 얼마인가?
17. UNIX 핵심부는 가상기억기에 있는 프로쎄스의 탄창을 필요에 따라 동적으로 증가시키지만 결코 그것을 줄이려고 하지 않는다. 어떤 프로그람이 10K 를 소비하는 탄창에 론리적인 배렬을 배정하는 C 부분루틴을 호출하는 경우를 고찰하자. 핵심부는 그것을 수용하기 위해 탄창토막을 확장한다. 부분루틴이 복귀할

때 탄창지시자를 조절하며 이 공간을 핵심부가 해방시킬수 있지만 해방되지 않는다. 이 점에서 탄창을 축소할수 있다는 리유와 UNIX 핵심부가 그것을 축소시키지 못하는 리유를 설명하시오.

18. LINUX 에서의 가상주소화방안을 보여 주는 그림 8-5 와 류사한 그림을 그리시오.

부록 8-ㄱ. 하쉬표

다음의 문제를 고찰하여 보자. N 개의 항목으로 된 모임이 어떤 표에 보관되여 있다. 매개 항목은 표식자와 그밖의 일정한 조건정보들로 구성되여 있다. 표에 대하여 삽입, 삭제 및 표식자에 의한 주어 진 항목의 탐색과 같은 몇가지 일반적인 조작을 하려고 한다.

항목들의 표식자가 $0 \sim M\text{-}1$ 범위에 있는 수자들이라면 단순한 풀이로서는 길이가 M 인 표를 사용하는것이다. 표식자 i 를 가진 항목은 위치 i 에서 그 표에 삽입된다. 항목들이 고정된 길이로 되여 있는한 표찾기는 어려우며 그 항목에 대한 수자식표식자에 기초하여 표에로 들어 가는 첨수화를 포함한다. 더우기 어떤 항목에 대한 표식자를 표안에 보관할 필요가 없다. 그것은 이것을 그 항목의 위치가 암시해 주고 있기때문이다. 그러한 표는 직접접근표로 알려 져 있다.

만일 표식자들이 비수자적인것이라도 여전히 직접접근방법을 사용할수 있다. 항목들을 A[1], A[N]으로 표시하기로 하자. 매개 항목 A[1]는 표식자 또는 열쇠 k_i 및 값 v_i 로 구성되여 있다. 사영함수 I(k)를 정의하자. I(k)는 모든 열쇠들에 대하여 1 과 M 사이의 값을 취하며 임의의 값 i 와 j 에 대하여 $I(ki) \neq I(kj)$ 이다. 이 경우에도 표의 길이가 M 과 같은 직접접근표를 사용할수 있다.

M 이 N 보다 훨씬 더 크다면 이 방안에서 한가지 난관이 발생한다. N 은 이 경우에 표에서 사용되지 않은 부분이 크고 기억기에 대한 효과적인 사용으로 되지 못한다. 택할수 있는 방안은 길이가 N 인 표를 사용하여 N 개의 항목들(표식자와 값)을 N 개의 표입구점에 기억시키는데 있을것이다. 이 방안에서 기억기의 량은 최소로 되지만 이제부터는 표를 찾기 위한 처리부담이 있다. 다음과 같은 가능성들이 있다. 즉

- **순차탐색 :** 열쇠전수공격방법은 표가 큰 경우에 시간을 소비한다.
- **련상탐색 :** 적당한 하드웨어를 사용하면 표안의 모든 요소를 동시에 탐색할수 있다. 이 방법이 일반목적이 아니여서 흥미 있는 임의의 그리고 모든 표들에 적용할수는 없다.
- **2 진탐색 :** 만일 표식자들이나 표식자에 대한 수자적인 사영을 표에서 올라 가는 순서로 배렬한다면 2 진탐색이 순차탐색(표 8-6)보다 훨씬 더 빠르며 그 어떤 특수한 하드웨어도 요구하지 않는다.

2 진탐색법은 표찾기에서 유망한것으로 되고 있다. 이 방법에서 주요한 결함은 새로운 항목추가가 보통 단순한 과정이 아니라는것이며 입구점들에 대한 재순서화를 요구한다는것이다. 따라서 2 진탐색법은 일반적으로 결코 변화되지 않는 적당한 정적인 표들에 대해서만 사용된다.

표 8-6. 길이가 M인 표에서 N개 항목중의 한개에 대한 평균탐색길이

수법	**탐색길이**
직접	1
순차	$(M+1)/2$
2 진	$\log_2 M$
선형하쉬법	$(2-N/M)/(2-2N/M)$
하쉬(련쇄 넘침)	$1+(N-1)/2M$

앞에서 렬거한 방법들중에서 단순한 직접접근법에 대한 기억기의 위반과 처리위반들을 피해야 한다. 이 타협안을 달성하는데서 가장 흔히 쓰이는 방법이 **하쉬법**이다. 1950년대에 개발된 하쉬법은 간단히 실현할수 있으며 두가지 우점을 가지고 있다. 우선 대부분의 항목들을 직접접근법에서와 같이 한번의 탐색으로 찾을수 있고 둘째로 추가적인 복잡성이 없이 삽입과 삭제를 처리할수 있다. 하쉬법기능을 다음과 같이 정의할수 있다.

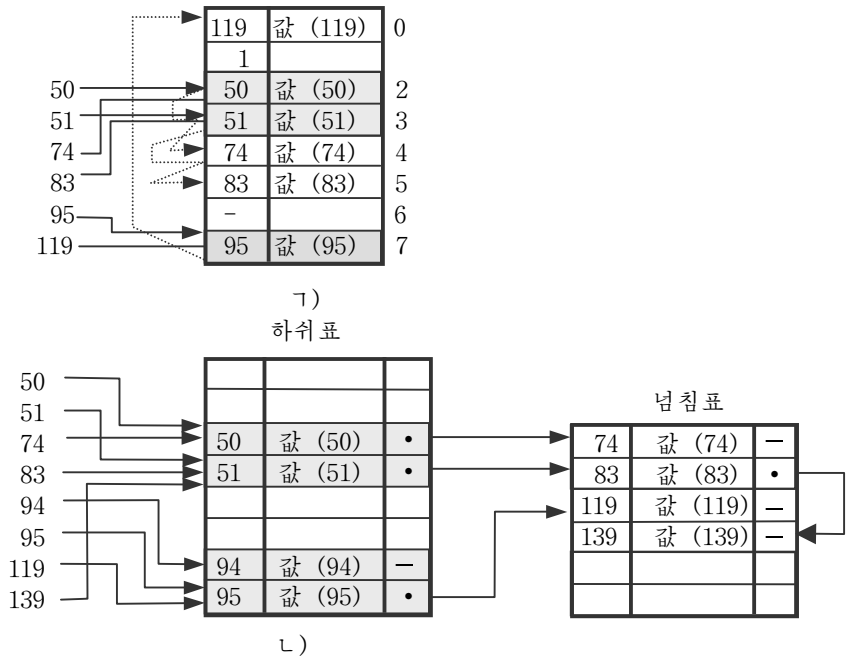



그림 8-26. 하쉬법
ㄱ-선형재하쉬법, ㄴ-련쇄

N 개까지의 항목이 길이가 M 인 하쉬표에 기억되여 있으며 $M \geq N$ 이나 N 보다 훨씬 크지는 않다고 하자. 표에 어떤 항목을 삽입하자면

I1. 항목의 표식자를 $0\sim M-1$ 사이에 있는 거의 우연적인 수 n 으로 변환한다. 실례로 표식자가 수자로 되여 있다면 일반적인 사영기능은 표식자를 M 으로 나누고 그 나머지를 n 의 값으로 취한다.

I2. n 을 하쉬표에 들어 가는 첨수로 사용한다.

ㄱ) 표에서 대응하는 입구점이 비여 있다면 항목(표식자와 값)을 입구점에 기억시킨다.

ㄴ) 입구점이 이미 차지되여 있으면 항목을 넘침구역에 이 부록에서 론의하는바 와 같이 기억시킨다.

표식자를 알고 있는 항목에 대한 표찾기를 수행하자면

L1. 삽입에서와 같은 사영기능을 사용하여 항목의 표식자를 $0 \sim M-1$ 사이에 있는 거의 우연적인 수 n으로 변환한다.

L2. n을 하쉬표에 들어 가는 첨수로 사용한다

ㄱ) 표에서 대응하는 입구점이 비여 있다면 항목은 이전에 표에 기억되여 있지 않았다.

ㄴ) 입구점이 이미 차지되여 있고 표식자가 맞는다면 값을 회복할수 있다.

ㄷ) 입구점이 이미 차지되여 있고 표식자가 맞지 않으면 넘침구역에서 탐색을 계속한다.

하쉬법방안은 넘침을 처리한다는데서 방법상 차이가 있다. 하나의 공통적인 수법을 선형하쉬법이라고 하는데 보통 콤파일러들에서 쓰인다. 이 방법에서 규칙 I2. ㄴ는

I2. ㄴ) 입구점이 이미 차지되여 있으면 $n = n + 1 \pmod M$으로 설정하고 단계 I2. ㄱ에로 돌아 간다.

와 같이 된다. 규칙 L2. ㄷ는 대응하게 변경된다.

그림 8-26 ㄱ가 하나의 실례로 된다. 이 경우에 기억되는 항목들의 표식자는 수자이며 하쉬표는 8개의 위치($M = 8$)를 가진다. 사영기능은 8로 나눈 나머지를 취한다. 그림은 항목들이 이것이 필요 없다고 하여도 올라 가는 수자순서로 삽입되였다고 가정하고 있다. 그러므로 항목 50과 51은 각각 위치 2와 3에로 사영하며 이것이 비여 있기때문에 그것들은 거기에 삽입된다. 위치 3을 시도한다. 이것역시 차지되여 있으므로 결국 위치 4를 사용한다.

클라스터를 이루는 효과가 있기때문에 열린 하쉬표에서 항목탐색의 평균길이를 결정하는것은 쉽지 않다. Schay와 Spruth[SCHA62]가 다음의 근사식을 얻었다. 즉

$$\text{평균탐색길이} = \frac{2-r}{2-2r} \quad \text{여기서} \quad r = N/M \text{ 이다.}$$

결과는 표의 크기에는 관계가 없고 다만 표가 얼마나 채워 져 있는가 하는데 관계된다는데 주목하자. 놀랄만한 결과는 표의 80%가 차 있을 때 평균탐색길이는 여전히 3 근방에 있다는것이다.

그렇다고 해도 탐색길이 3이 길다고 고찰할수 있는데 선형하쉬법의 표는 항목들을 삭제하기 쉽지 않다는 추가적인 문제를 가지고 있다. 탐색길이가 보다 짧고(표 8-6) 삭제는 물론 추가도 할수 있는 좋은 방법이 바로 련쇄넘침법이다. 이 수법을 그림 8-26 ㄴ에서 설명하고 있다. 이 경우에는 넘침입구점을 삽입하는 개별적인 표가 있다. 이 표는 하쉬표에서 임의의 위치와 관련되여 있는 련쇄입구점들을 넘겨 주는 지시자들을 포함한다. 이 경우에 우연적으로 분포된 자료라고 가정하면 평균탐색길이는

$$\text{평균탐색길이} = 1 + (N-1) / 2M \quad \text{이다.}$$

큰 N 및 M값에 대하여 이 값은 $N = M$일 때 1.5에 가까이 간다. 그러므로 이 수법은 고속찾기를 가진 밀집기억을 허락한다.

제 4 편. 일정작성

제 4 편의 중심

조작체계는 경쟁하는 여러 프로쎄스들의 요구들사이에서 콤퓨터에 자원들을 배정해야 한다. 처리기의 경우에 배정해야 할 자원은 처리기상에서의 집행시간이며 배정수단은 일정작성이다. 일정작성기능은 공평성, 임의의 특정한 프로쎄스의 고갈결핍, 처리기시간의 효과적사용 및 적은 간접소비시간을 포함하는 많은 대상들을 만족시키도록 설계되여야 한다. 또한 일정작성기능은 일정한 프로쎄스의 시작이나 완료에서 각이한 준위의 우선권이나 실시간적인 기한부들을 고려할수도 있다.

여러 해동안 일정작성은 많은 연구의 초점으로 되여 왔으며 많은 각이한 알고리듬들이 실현되였다. 오늘날 일정작성연구에서 강조할 문제는 다중처리기체계 특히 다중스레드식응용프로그람개발과 실시간일정작성에 대한것이다.

제 4 편의 안내

제 9 장. 단일처리기의 일정작성

제9장은 단일처리기를 가진 체계에서의 일정작성에 관한것이다. 이 제한된 지면상에서 일정작성과 관련된 많은 설계문제들을 정의하고 해명한다는것은 불가능하다. 제9장에서는 세가지 형태의 처리기일정작성 즉 장기, 중기, 단기일정작성에 대한 고찰로부터 시작한다. 이 장의 대부분은 단기일정작성문제에 중심을 두고 있다. 여기서는 각이한 알고리듬들을 고찰하고 특성들을 비교한다.

제 10 장. 다중처리기와 실시간일정작성

제10장에서는 일정작성연구의 초점인 두 령역을 고찰한다. 여러개의 처리기의 존재는 일정작성결정을 복잡하게 만들며 새로운 기회들을 제공한다. 특히 여러개의 처리기를 사용하면 동일한 프로쎄스안에서 여러개의 스레드를 집행하기 위한 동시일정작성을 할수 있다. 제10장의 첫 부분에서는 다중처리기 및 다중스레드식일정작성법들을 개괄한다. 장의 나머지부분에서는 실시간일정작성법들을 취급한다. 실시간요구사항은 일정작성기가 만족시켜야 할 가장 중요한 요구인데 그것은 요구사항들이 주어 진 과제나 프로쎄스의 시작과 완료에서의 시간제한을 규정해 줌으로써 공평성이나 우선권을 넘어 서기때문이다.

제 9 장. 단일처리기의 일정작성

다중프로그람처리체계에서 다중프로쎄스들은 주기억기에 있다. 매개 프로쎄스들은 처리기를 사용하거나 입출력이 수행되거나 어떤 다른 사건이 발생되기를 기다린다. 하나의 처리기나 여러 처리기들은 한개의 프로쎄스집행으로 차지되며 다른것들은 기다린다. 다중프로그람처리에서 기본은 일정작성이다. 일정작성법의 형태에는 대표적으로 네가지가 있다(표 9-1). 그것들중의 하나인 입출력일정작성은 입출력을 취급하는 제11장에서 설명한다. 처리기의 일정작성의 형태인 나머지 세가지 형태는 이 장과 다음장에서 설명한다.

이 장에서는 먼저 세가지 형태의 처리기일정작성을 고찰하며 관계되는 문제를 설명한다. 장기일정작성과 중기일정작성이 다중프로그람처리의 등급과 관계되는 성능에 의해 좌우된다는것을 보게 된다. 이 문제는 제3장에서 어느 정도 취급하였고 제7장과 제8장에서 좀더 구체적으로 취급하였다. 그리고 이 장의 나머지부분에서는 단기일정작성에 대하여 집중적으로 고찰하며 그것은 단일처리기체계에 대한 일정작성의 고찰로 제한한다. 다중처리기의 사용은 보충적인 복잡성을 더해 주므로 먼저 단일처리기의 경우에로 집중하는것이 좋다. 그래야 일정작성알고리듬사이의 차이점들을 명백히 알수 있다.

제2절에서는 단기일정작성을 결정하는데 사용할수 있는 여러가지 알고리듬들을 보게 된다.

제 1 절. 일정작성의 형태

처리기일정작성의 목적은 응답시간, 처리능력, 처리기효률과 같은 체계의 목표를 달성하는데서 한동안 하나의 처리기나 여러 처리기들에 의하여 프로쎄스들이 실행되도록 할당하는것이다. 많은 체계들에서 일정작성작업은 세가지 단독적인 기능 즉 장기, 중기 및 단기일정작성으로 분해된다. 이름은 이 기능들을 수행하는 상대적인 시간척도를 암시해 준다.

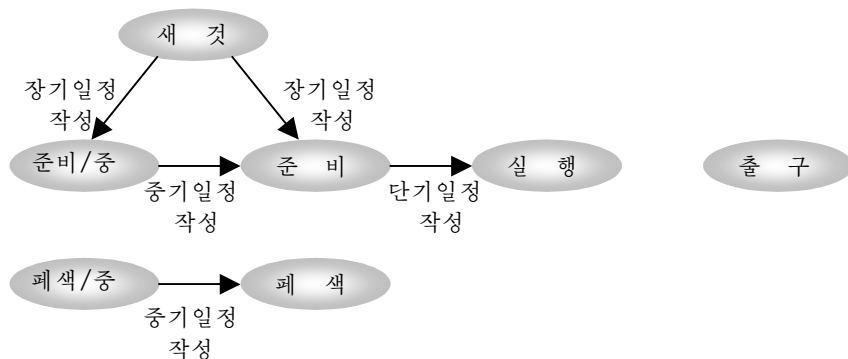



그림 9-1. 일정작성과 프로쎄스상태이행

그림 9-1에서는 일정작성기능들을 프로쎄스상태이행도(그림 3-8에 제시됨)로 보여 주고 있다. 장기일정작성은 새로운 프로쎄스가 창조될 때 수행된다. 이것은 현재 작업하고 있는 프로쎄스모임에 새로운 프로쎄스를 합치기 위한 결정이다. 중기일정작성은 기억기교체기능의 일부이다. 이것은 주기억기에 적어도 부분적으로 있고 따라서 수행에 사용할수 있는것에 어떤 프로쎄스를 합치기 위한 결정이다. 단기일정작성은 프로쎄스가 다음에 집행

표 9-1. 일정작성법의 형태

장기일정작성	실행하여야 할 프로세스들의 집결소를 증가시키는 결정
중기일정작성	주기억기에 부분적으로 또는 통채로 있는 프로세스들의 수를 증가시키는 결정
단기일정작성	사용가능한 프로세스를 처리기에 의하여 실행하는 결정
입출력일정작성	프로세스의 긴박한 입출력요청을 사용가능한 입출력장치에 의하여 처리시키는 결정

할수 있도록 준비하는 실제적인 결정이다. 그림 9-2는 일정작성기능의 겹침을 보여 주기 위하여 그림 3-8ㄴ의 상태이행도를 다시 작성한것이다.

일정작성은 프로세스를 기다리거나 진행하는것을 결정하기때문에 체계의 성능에 영향을 준다. 이것을 그림 9-3에 제시하였는데 프로세스의 상태이행들에 동반되는 대기렬을 보여 주고 있다.[1] 본질상 일정작성은 대기렬짓기지연을 최소화하고 대기렬환경에서 성능을 최적화하기 위하여 대기렬이동을 관리하는 문제이다.

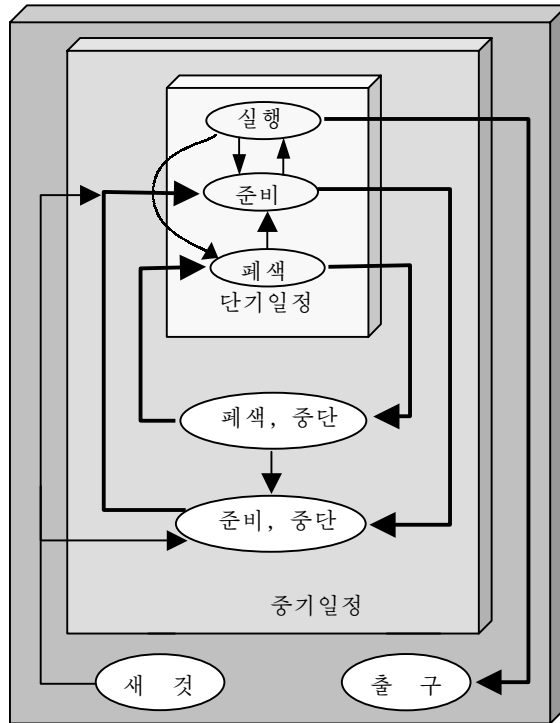



그림 9-2. 일정작성의 준위

장기일정작성

[1] 간단히 하기 위하여 그림 9-3에서는 준비상태에 직접 들어 가는 새로운 프로세스를 보여 주고 그림 9-1, 9-2에서는 준비상태 또는 준비/중단상태를 보여 주고 있다.

장기일정작성기는 처리를 위해 프로그람들이 체계에 입장하는것을 결정한다. 이때 그것은 다중프로그람처리의 등급을 조종한다. 일단 입장이 허락되면 일감이나 사용자프로그람은 프로쎄스로 되며 단기일정작성기의 대기렬에 보충된다. 일부 체계들에서는 새롭게 창조된 프로쎄스가 교체내기조건에서 시작되는데 그러한 경우에 그것은 중기일정작성기의 대기렬에 보충된다.

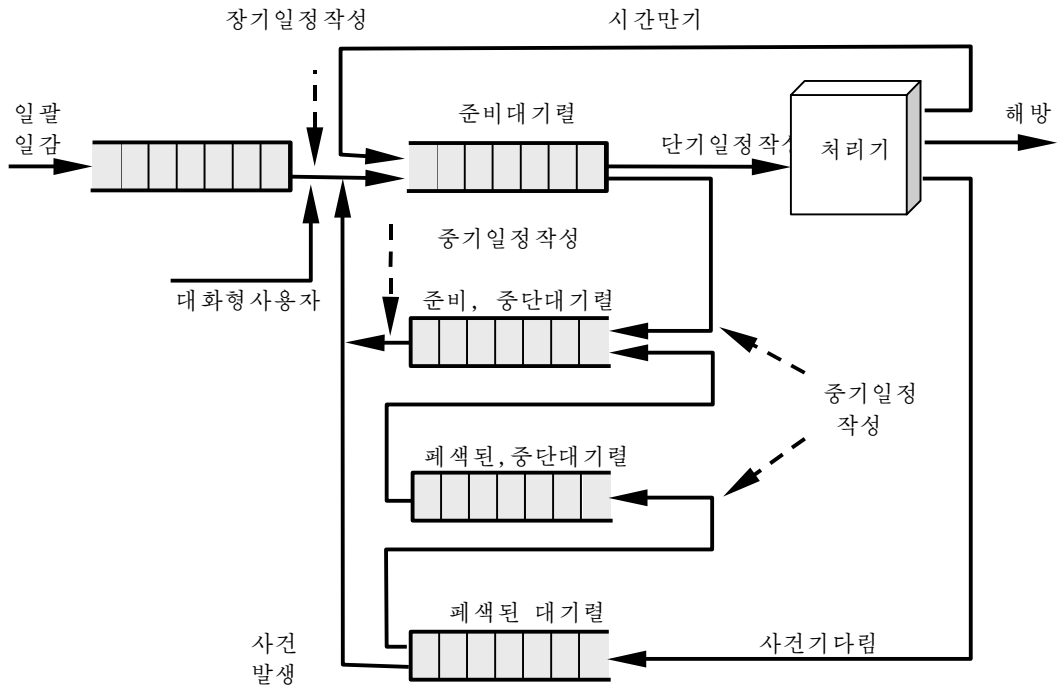



그림 9-3. 일정작성을 위한 대기렬작성도식

일괄체계에서 또는 일반조작체계의 일괄부분에서 새롭게 제기된 일감들은 디스크에로 발송되고 일괄대기렬에 들어 선다. 장기일정작성기에서는 그것이 할수 있을 때 대기렬로부터 프로쎄스를 창조한다. 여기서 제기되는 두가지 결정이 있다. 첫째로, 일정작성기는 조작체계가 한개 또는 그이상의 보충적인 프로쎄스들을 받아 들일수 있는가를 결정해야 한다. 둘째로, 일정작성기가 일감 또는 일감들을 접수하고 프로쎄스에로 방향을 돌리겠는가를 결정하여야 한다. 이러한 두가지 결정문제를 간단히 고찰하자.

새로운 프로쎄스를 언제 창조하겠는가에 대한 결정은 일반적으로 다중프로그람처리기가 요구하는 등급에 의하여 좌우된다. 창조되는 프로쎄스가 많을수록 매개 프로쎄스가 집행할수 있는 시간의 백분률은 더 작아 진다(즉 많은 프로쎄스들이 동일한 크기의 처리시간에 경쟁을 한다.). 이와 같이 장기일정작성기는 현재 프로쎄스모임에 대한 봉사를 충분히 하기 위하여 다중프로그람처리기의 등급을 제한할수 있다. 일감을 끝낼 때마다 매번 일정작성기는 한개 또는 그이상의 새로운 일감들을 보충하도록 결정할수 있다. 보충적으로 처리기가 놀고 있는 시간이 어떤 턱값을 넘을 때에는 장기일정작성기를 요청할수 있다.

일감을 다음에 입장시키겠는가에 대한 결정은 단순한 선래선봉사의 원칙에서 할수 있다. 또는 어떤 도구가 체계성능을 관리하게 할수 있다. 이때 사용하는 기준에는 우선권, 예상되는 집행시간 그리고 입출력요구사항을 포함할수 있다. 실례로 정보가 사용가능하면 일

정작성기는 처리기위주 및 입출력위주의 프로쎄스[2]들을 혼합하여 유지하려고 할수 있다. 또한 결정은 입출력사용의 균형을 맞추기 위한것으로서 입출력자원들을 요구하려고 하는가에 따라 내릴수 있다.

시분할체계의 대화형프로그람들에서 프로쎄스요청은 체계에 접속하려는 사용자의 시도에 의하여 발생될수 있다. 시분할사용자들은 체계가 자기들을 접수할수 있을 때까지 단순히 대기렬을 짓고 기다리려고 하지 않는다. 오히려 조작체계는 미리 정의한 어떤 포화측정기구를 사용하여 체계가 포화될 때까지 허가된 모든 오는것들을 접수한다. 그때 접속요청은 체계가 충만되여 사용자가 후에 다시 오라는것을 표시하는 통보를 만나게 된다.

중기일정작성

중기일정작성은 교체기능의 일부이다. 관련된 문제들을 제3장, 제7장, 제8장에서 취급하였다. 대표적으로 교체넣기결정은 다중프로그람처리의 등급을 관리할 필요에 따라 진행한다. 가상기억기를 사용하지 않는 체계에서는 기억기관리가 또한 문제이다. 그러므로 교체결정은 교체내기프로쎄스들의 기억기요구사항을 고려한다.

단기일정작성

실행빈도의 측면에서 보면 장기일정작성기는 상대적으로 드물게 실행되며 새로운 프로쎄스를 취하겠는지 안하겠는지 또 한개를 취하겠는지 하는 굵은알갱이식의 결정을 한다. 중기일정작성기는 교체결정을 하기 위하여 얼마간 더 자주 실행되나 배분기라고도 하는 단기일정작성기는 자주 실행되는데 프로쎄스를 다음에 실행하겠는가 하는 가는알갱이식의 결정을 한다.

단기일정작성프로그람은 사건이 발생할 때마다 요청되는데 이것은 현재 프로쎄스의 중단에로 이끌어 갈수 있거나 다른 프로쎄스를 위하여 현재 실행중에 있는 프로쎄스를 선취할수 있는 기회를 준다. 실례로 그러한 사건으로서는 다음의것들이 있다. 즉

- 박자새치기
- 입출력새치기
- 조작체계호출
- 신호

제 2 절. 일정작성알고리듬

단기일정작성기준

단기일정작성의 기본목표는 체계의 동작을 여러가지 각도에서 최적화하는 방향으로 처리시간을 배정하는것이다. 일반적으로 기준모임은 각이한 일정작성방법을 평가할수 있다는것을 고려하여 제정한다.

공통적으로 사용하는 기준들을 두개의 차원으로 분류할수 있다. 우선 사용자지향과 체계지향의 기준들을 제정할수 있다. 사용자지향의 기준들은 개별적인 사용자나 프로쎄스로리해하는 체계의 동작과 관련된다. 실례로서 대화형체계의 응답시간을 들수 있다. 응답시

[2] 프로쎄스가 주로 계산작업을 수행하고 때때로 입출력장치를 사용하면 그것을 처리기위주프로쎄스로, 처리기를 사용하는 시간보다 입출력장치를 사용하는 시간이 많은 프로쎄스를 입출력위주프로쎄스라고 한다.

간은 요청을 해서부터 응답이 출력으로 나타나기 시작할 때까지 경과한 시간이다. 이 량은 사용자가 변경할수 있고 또 본래 사용자가 관심하는 량이다. 각이한 사용자들에게 《좋은》봉사를 주는 일정작성을 하려고 한다. 응답시간인 경우에 턱값은 실례로 2s로 정의할수 있다. 그러면 일정작성기구의 목표는 평균응답시간이 2s이거나 그보다 적은 시간을 체험한 사용자의 수를 최대로 하는것이다.

다른 기준들은 체계지향형이다. 기본은 처리기의 유효하고도 능률적인 사용에 있다. 실례로 프로쎄스들이 완료되는 속도인 처리능력이다. 이것은 체계성능을 표시하는 훌륭한 측정지표로서 최대화하고 싶은 량이다. 그러나 그것은 사용자에게 주는 봉사보다도 체계의 성능에 더 집중한다. 이와 같이 그것은 사용자집단이 아니라 체계관리에 관계되는것이다.

한편 사용자지향의 기준들은 가상적으로 모든 체계들에 중요하지만 체계지향의 기준들은 일반적으로 단일사용자체계들에는 그리 중요하지 않다. 단일사용자체계에서 사용자응용프로그람에 대한 체계의 응답을 접수하는 동안 높은 처리기사용률과 높은 처리능력을 달성하는것은 그리 중요하지 않다.

기준을 분류할수 있는 다른 차원은 성능에 관계되는것과 성능에 직접 관계되지 않는것이다. 성능에 관계되는 기준들은 정량적인것들로서 쉽게 측정할수 있다. 실례로서 응답시간과 처리능력을 들수 있다. 성능과 관계되지 않는 기능들은 사실상 정상적이거나 측정과 분석을 쉽게 할수 없는것들이다. 그러한 기준의 실례로서 예측을 들수 있다. 사용자에게 주는 봉사가 체계로 수행되는 다른 작업에는 관계없이 동일한 특성들을 발휘하는것이 중요하다. 이 기준은 작업적재에 대한 함수로서 변동의 정도를 계산하는 방법으로 다소 측정할수 있다. 그러나 작업적재함수로서 처리능력이나 응답시간을 측정하는것은 간단한 문제가 아니다.

표 9-2에는 중요한 일정작성기준을 요약하였다. 이것들은 호상 의존관계에 있고 따라서 그 모든것들을 동시에 최적화할수는 없다. 실례로 좋은 응답시간을 보장하자면 프로쎄스들을 자주 전환하는 일정작성알고리듬이 필요하다. 이것은 체계의 간접소비시간을 증가시켜 처리능력을 감소시킨다. 따라서 일정작성방법을 설계하는데서는 상반되는 요구들의 타협이 이루어 지게 한다. 즉 각이한 요구사항들에 부여한 상대적인 무게들은 체계의 성질과 사용에 관계된다.

많은 대화형조작체계들에서는 단일사용자이든 아니면 시분할이든 적당한 응답시간이 주요한 요구로 된다. 이 요구사항이 중요하고 또한 응용에 따라 그것에 대한 정확한 정의가 달라 지므로 이 문제에 대해서는 이 장의 부록에서 좀 더 구체적으로 설명하기로 한다.

우선권의 사용

많은 체계에서 매개 프로쎄스에는 우선권이 할당되며 일정작성프로그람은 항상 우선권이 낮은것보다 우선권이 더 높은 프로쎄스를 선택한다. 그림 9-4에서는 우선권의 사용을 보여 주고 있다. 명백히 하기 위하여 다중폐색된 대기렬들과 중단된 상태들은 무시하고 대기렬짓기도식을 간단화하였다(그림3-7과 비교하시오.). 한개의 준비대기렬대신에 우선권이 낮아 지는 순서로 된 대기렬모임을 준다. 즉 $i<j$에 대하여 RQ0, RQ1,…RQn인 경우에 [RQi]의 우선권>[RQj]의 우선권이다.[3] 일정작성선택을 하려고 할 때 일정작성기는 우선권이 제일 높은 준비대기렬(RQ0)에서 시작된다. 만일 대기렬에 한개이상의 프로쎄스들이 있다면 일정작성원칙을 적용하여 그중에서 한개의 프로쎄스를 선택한다. 만일 RQ0이 비였다면 RQ1을 조사한다. 이러한 과정이 계속된다.

순수한 우선권일정작성도식과 관련하여 제기되는 문제는 우선권이 낮은 프로쎄스들이

---

[3] UNIX와 다른 체계들에서 우선권값이 클수록 우선권이 더 낮은 프로쎄스를 표시한다. 다른 지적이 없는한 이러한 관례에 따르도록 한다. 즉 번호가 클수록 우선권순위가 더 높다.

고갈을 당할수 있는것이다. 우선권이 높은 준비프로쎄스들이 항상 안전하게 공급될 때 고갈이 일어 난다. 만일 이 동작을 바라지 않는다면 프로쎄스의 우선권을 그것의 발생나이나 실행경력에 의해 변화시킬수 있다. 이것에 대한 실례를 아래에서 보기로 한다.

여러가지 일정작성방책

이 소절에서는 심의하는 각이한 일정작성방책론에 대한 일부 요약된 자료를 표 9-3에 제시하였다. 그 기능은 우선권, 자원요구사항이나 프로쎄스의 실행특성지표에 기초할수 있다. 후자의 경우에 세가지 량이 중요하다. 즉

표 9-2. 일정작성기준

	사용자지향 및 성능지향
일감처리시간	이것은 프로쎄스의 의뢰와 그것의 완료사이의 시간간격이다. 실제적인 실행시간에 처리기를 포함하여 자원들을 기다리는데 소비한 시간을 더한것이다. 이것은 일괄일감에 알맞는 측정지표이다.
응답시간	대화형프로쎄스에서 요청이 의뢰된 순간부터 응답이 수신되기 시작한 순간까지의 시간이다. 흔히 요청을 계속 처리하는동안 프로쎄스는 사용자에게 출력을 낼수 있다. 그러면 이것은 사용자의 견지에서 일감처리시간보다 더 좋은 측정지표로 된다. 일정작성에서는 응답시간을 적게 하면서 만족스러운 응답시간을 보장 받는 대화사용자를 최대화하기 위해 노력해야 한다.
기한부	프로쎄스완료기한부를 명시할수 있는 경우에 일정작성규칙은 다른 목표들을 기한부만족률을 최대화하는데 복종시켜야 한다.
	사용자지향, 기타
예측가능성	주어 진 일감은 체계우의 적재에는 관계없이 대략 같은 크기의 시간에 그리고 동일한 가격으로 실행되여야 한다. 응답시간이나 일감처리시간의 변동이 크면 사용자들에게 혼란을 가져다 준다. 그것은 체계의 작업적재시 일어나는 심한 변동에 대하여 알려 주거나 체계가 불안정요소들을 제거하는데로 넘어 가는것을 경고한다.
	체계지향, 성능지향
처리능력	일정작성방법에서는 단위시간당 완료되는 프로쎄스를 최대화하려고 해야 한다. 이것은 얼마만한 작업을 수행하고 있는가를 표시하는 측정지표이다. 이것은 명백히 프로쎄스의 평균길이에 의존되면서도 일정작성방법의 영향을 받는다. 처리능력은 사용률에 영향을 미칠수 있다.
프로쎄스사용률	이것은 처리기가 점유하는 시간의 백분률이다. 값비싼 공유체계에서는 주요한 기준으로 된다. 단일사용자체계와 실시간체계와 같은 일부 다른 체계들에서는 이것이 다른 기준들보다는 그리 중요하지 않다.
	체계지향 ,기타
공평성	사용자의 안내나 다른 체계에 의한 안내가 없는 경우에 프로쎄스들은 동일하게 취급된다. 이때에는 프로쎄스들이 고갈을 당하지 않는다.
우선권시행	프로쎄스들에 우선권이 할당되였을 때 일정작성에서는 우선권이 더 높은 프로쎄스들을 특별히 고대한다.
자원균형	일정작성방법은 체계자원이 계속 차지상태에 있게 한다. 강조된 자원들을 충분히 사용하지 않는 프로쎄스들은 특별히 고려한다. 이 기준은 중기 및 단기일정작성에 관계된다.

w = 체계에서 기다림과 실행에 의해 지금까지 소비한 시간

e = 지금까지 소비한 시간

s = e를 포함하여 프로쎄스에 요구되는 총 봉사시간, 일반적으로 이 량은 추산하거나 사용자에 의해 제기되여야 한다.

실례로 선택기능 max[w]는 선래선봉사(FCFS)규칙을 표시한다.

결정방식은 선택기능이 발휘되는 순간을 표시한다. 여기에는 일반적으로 두가지 부류가 있다. 즉

- **비선취 :** 이 경우에 프로쎄스가 일단 실행상태에 들어 가면 프로쎄스는 ㄱ) 그것이 끝날 때까지 계속 실행되며 ㄴ) 입출력을 기다리거나 조작체계봉사를 요청하기 위하여 그 자체를 페색한다.

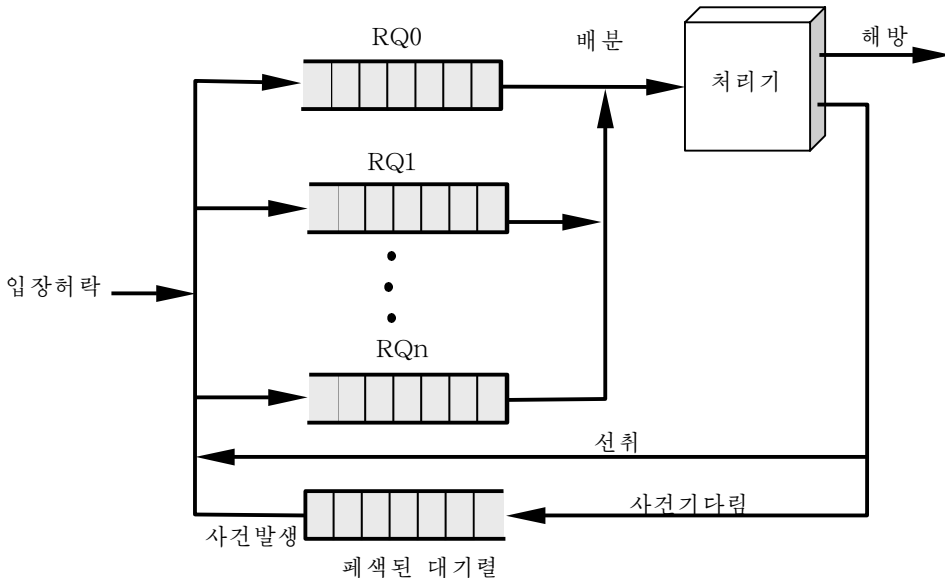



그림 9-4. 우선권대기렬짓기

- **선취:** 현재 실행중에 있는 프로쎄스는 새치기되여 조작체계에 의하여 준비상태에로 이행할수 있다. 선취에 대한 결정을 새로운 프로쎄스가 도착할 때, 페색된 프로쎄스를 준비상태에 배치하는 새치기가 일어 나거나 박자에 의한 새치기가 주기적으로 발생할 때 하게 된다.

선취방책들은 비선취방책들에서보다 더 큰 간접소비시간을 초래하지만 총체적인 프로쎄스집단에는 더 좋은 봉사를 줄수 있다. 왜냐하면 그것들은 임의의 프로쎄스가 오래동안 처리기를 독점하는것을 막을수 있기때문이다. 게다가 효과적인 프로쎄스절환기를 사용(하드웨어로부터 가능한 많은 방조를 받는다.)하고 주기억기에 프로그람의 많은 몫을 보존하도록 큰 주기억기를 줌으로써 선취한 방식의 가격은 상대적으로 낮게 유지할수 있다.

각이한 일정작성방책을 서술하면서 실행중에 있는 프로쎄스모임을 표 9-4에 제시하였다. 여기서 필요한 총 실행시간을 봉사시간으로 보고 그것들을 일괄일감들로 생각할수 있다. 한편 그것들을 처리기와 입출력을 반복하여 번갈아 사용하는것을 필요로 하는 진행프로쎄스라고도 볼수 있다. 후자의 경우에 봉사기간은 한개의 주기동안에 필요한 처리기시간을 표시한다. 두 경우에 대기렬모형에 의하면 이 량은 봉사시간[4]에 해당한다.

---

4. 대기렬모형의 용어들이 요약되여 있는 부록 9－ㄴ를 보라.

표 9-3. 각이한 일정작성방책의 특성지표

방책	선택기능	결정방식	처리능력	응답시간	간접소비시간	프로쎄스에 대한 효과	고갈
FCFS법	max[w]	비선취	강조안됨	특히 프로쎄스실행시간이 크게 변동되는 경우에 클 수 있다.	최소	짧은 프로쎄스들에 벌칙을 준다; 입출력위주프로쎄스들에 벌칙을 준다.	없음
순환법	일정	선취(시간량자에서)	량자가 지내 작으면 낮을 수 있다.	짧은 프로쎄스에 대하여 좋은 응답시간을 보장한다.	최소	취급이 순조롭다.	없음
SPN법	min[s]	비선취	높다.	짧은 프로쎄스에 대하여 좋은 응답시간을 보장한다.	클수 있다.	긴 프로쎄스들에 벌칙을 준다.	가능
SRT법	min[s-e]	선취(도착에서)	높다.	좋은 응답시간을 보장한다.	클수 있다.	긴 프로쎄스들에 벌칙을 준다.	가능
HRRN법	max($\frac{w+s}{s}$)	비선취	높다.	좋은 응답시간을 보장한다.	클수 있다.	좋은 균형을 보장한다.	없음
반결합법	(책을 보라)	선취(시간량자에서)	강조안됨	강조 안됨	클수 있다.	입출력위주프로쎄스들에 편중할 수 있다.	가능

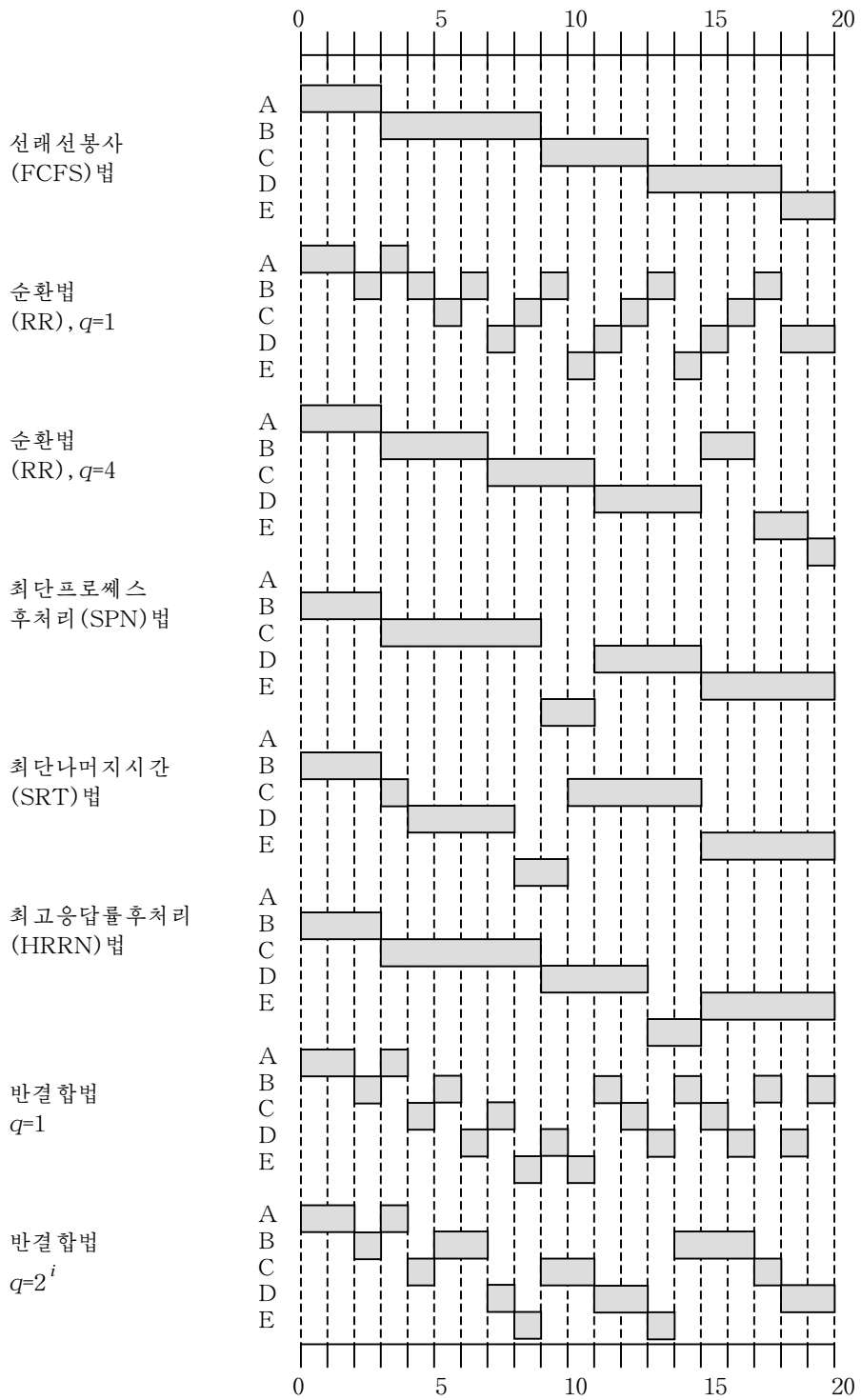



그림 9-5. 일정작성방책의 비교

표 9-4. 프로쎄스일정작성의 실례

프로쎄스	도착시간	봉사시간
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

선래선봉사법

가장 간단한 일정작성방책은 선입선출(FIFO)법과 엄밀한 대기렬모형으로 알려 진 선래선봉사(FCFS)법이다. 매개 프로쎄스가 준비상태에 있을 때 그것은 준비대기렬을 련결시킨다. 현재 실행중에 있는 프로쎄스가 실행을 멈추었을 때 준비대기렬에 있던 프로쎄스가 선택되여 실행된다.

그림 9-5에서는 지적한 실례에 대한 한 주기동안의 실행패턴을 보여 주고 있고 표 9-5에는 일부 중요한 결과들을 제시하고 있다. 우선 매개 프로쎄스의 최종시간이 확정되면 이로부터 일감처리시간을 결정할수 있다. 대기렬모형에 의하여 **일감처리기시간**(TAT)은 상주시간 T_r 또는 항목이 체계에서 소비한 총 시간(기다림시간 더하기 봉사시간)이다. 보다 쓸모 있는 값은 정규화된 일감처리시간 즉 봉사시간에 대한 일감처리시간의 비이다. 대표적으로 실행시간이 길수록 허용할수 있는 절대적인 지연량은 더 커진다. 이 비률의 가능한 최소값은 1.0이다. 이 값의 증가는 감소하는 봉사준위에 대응한다.

FCFS법은 짧은 프로쎄스들보다 긴 프로쎄스들에 훨씬 더 잘 맞는다. [FINK88]의 자료에 기초한 다음의 실례를 고찰하자. 즉

프로쎄스	도착시간	봉사시간(T_s)	시작시간	마감시간	일감처리시간T_r	T_r/T_s
W	0	1	0	1	1	1
X	1	00	1	101	100	1
Y	2	1	101	102	100	100
Z	3	00	102	202	199	1.99
평균					100	26

프로쎄스 Y에 대한 정규화된 일감처리시간은 다른 프로쎄스들에 비하여 한자리 더 크다. 체계에서 소비되는 총 시간은 요구되는 처리시간의 100배이다. 이것은 긴 프로쎄스 바로 뒤에 짧은 프로쎄스가 도착할 때마다 일어난다. 한편 이 극단한 실례에서조차 긴 프로쎄스는 형편이 나쁘지 않다. 프로쎄스 Z의 일감처리시간은 Y것의 두배이고 그것의 정규화된 기다림시간은 2.0이하이다.

FCFS법과 관련된 다른 곤난성은 그것이 입출력위주의 프로쎄스들보다 처리기위주의 프로쎄스들에 편중되는 경향이 있는것이다. 처리기(처리기위주)를 제일 많이 사용하는 한개의 프로쎄스와 입출력(입출력위주)을 기본으로 하는 많은 프로쎄스들을 포함한 프로쎄스집단을 고찰하자. 처리기위주의 프로쎄스가 실행중에 있을 때 모든 입출력위주의 프로쎄스들은 기다려야 한다.이것들중에서 일부는 입출력대기렬(폐색된 상태)에 있을수 있으나 처리기위주의 프로쎄스가 실행중에 있는동안 준비대기렬에로 도로 이행할수 있다. 이 시점에서 대부분의 또는 모든 입출력장치들은 작업할 능력이 있음에도 불구하고 노는 상태에 있을수 있다. 현재 실행중에 있는 프로쎄스가 실행상태에서 리탈할 때 준비된 입출력위주의 프로쎄스들은 즉시에 실행상태에로 이동하여 입출력사건들을 폐색한다. 만일 처리

기위주의 프로쎄스도 폐색되여 있다면 그것도 노는 상태에 있다. 그리하여 FCFS법은 결과적으로 처리기와 입출력장치들을 둘다 효률적으로 사용하지 못하게 할수 있다.

FCFS법은 단일처리기체계에서 적용할수 있는 좋은 방법이 못된다. 이로부터 흔히 우선권방안과 조합하여 효과적인 일정작성기를 보장하고 있다. 그리하여 일정작성기는 매개 우선권준위에 한개의 대기렬을 할당시킨 많은 대기렬을 유지하면서 매개 대기렬안에서는 선래선봉사방법에 기초하여 배분할수 있다. 그러한 체계의 실례는 뒤에서 반결합일정작성법을 취급할 때 보게 된다.

표 9-5. 일정작성방책들의 비교

							평균
	프로쎄스	A	B	C	D	E	
	도착시간	0	2	4	6	8	
	봉사시간(T_s)	3	6	4	5	2	
FCFS	마감시간	3	9	13	18	20	
	일감처리시간(T_r)	3	7	9	12	12	8.60
	T_r / T_s	1.00	1.17	2.25	2.40	6.00	2.56
RR	마감시간	4	18	17	20	15	
q=1	일감처리시간(T_r)	4	16	13	14	7	10.80
	T_r / T_s	1.33	2.67	3.25	2.80	3.50	2.71
RR	마감시간	3	17	11	20	19	
q=4	일감처리시간(T_r)	3	15	7	14	11	10.00
	T_r / T_s	1.00	2.5	1.75	2.80	5.50	2.71
SPN법	마감시간	3	9	15	20	11	
	일감처리시간(T_r)	3	7	11	14	3	7.60
	T_r / T_s	1.00	1.17	2.75	2.80	1.50	1.84
SRT법	마감시간	3	15	8	20	10	
	일감처리시간(T_r)	3	13	4	14	2	7.20
	T_r / T_s	1.00	2.17	1.00	2.80	1.00	1.59
HRRN	마감시간	3	9	13	20	15	
법	일감처리시간(T_r)	3	7	9	14	7	8.00
	T_r / T_s	1.00	1.17	2.25	2.80	3.5	2.14
FB q=1	마감시간	4	20	16	19	11	
	일감처리시간(T_r)	4	18	12	13	3	10.00
	T_r / T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB	마감시간	4	17	18	20	14	
$q=2^i$	일감처리시간(T_r)	4	15	14	14	6	10.60
	T_r / T_s	1.33	2.50	3.50	2.80	3.00	2.63

순환법

짧은 일감들이 FCFS법에서 당하는 벌칙을 감소시키는 간단한 방법은 박자에 기초한 선취권을 사용하는것이다. 박자새치기는 주기적인 시간간격으로 발생된다. 새치기가 발생할 때 현재 실행중에 있는 프로쎄스는 준비대기렬에 배치되며 다음의 준비일감은 FCFS법으로 선택된다. 이 수법을 시간세분법이라고도 하는데 그것은 매개 프로쎄스가 선취하기전에 세분된 시간을 할당 받기때문이다.

순환법에서 기본설계문제는 시간량자가 대단히 짧기때문에 짧은 프로쎄스들이 체계를 걸쳐 상대적으로 빨리 이동한다. 한편 박자새치기를 처리하고 일정작성 및 배분기능을 수행하는

데 걸리는 처리간접소비시간이 있다. 대단히 짧은 시간량자는 피하여야 한다. 한가지 효과적인 방법은 시간량자를 표준적인 상호작용에 필요한 시간보다 조금 크게 하는것이다. 만일 그것이 작다면 대부분의 프로쎄스들은 적어도 두배의 량자를 요구하게 된다. 그림 9-6에서는 이

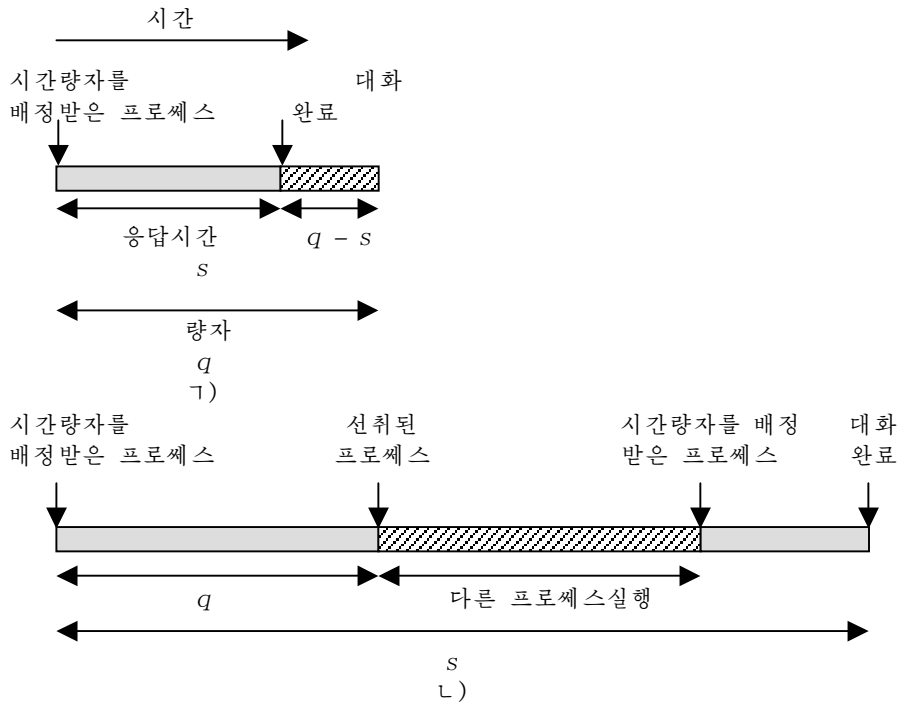



그림 9-6. 선취시간량자의 크기의 효과: ㄱ-표준적인 상호작용보다 큰 시간량자, ㄴ-표준적인 상호작용보다 작은 시간량자

것이 응답시간에 미치는 효과를 보여 주고 있다. 시간량자가 가장 긴 실행중의 프로쎄스보다 더 긴 극단한 경우에 순환법은 FCFS법에로 퇴보한다는것을 주의해야 한다.

그림 9-5와 표 9-5에서는 각각 1배와 4배 단위의 시간량자 q를 사용하는 실례에 대한 결과를 보여 주고 있다. 가장 짧은 일감인 프로쎄스 E는 시간량자가 1인 경우에 현저한 개선을 가져 온다는것을 알수 있다.

순환법은 일반 시분할체계나 트랜잭션처리체계에서 특히 효과적이다. 순환법의 한가지 결함은 처리기위주 및 입출력위주의 프로쎄스보다 더 짧은 처리기시간단락(입출력연산들사이의 실행에 소비되는 시간량)을 가진다. 만일 처리기위주 및 입출력위주의 프로쎄스들이 혼합되여 있다면 다음의것이 발생된다. 입출력의 프로쎄스는 짧은 시간동안 처리기를 사용하며 입출력에서 페색된다. 그것은 입출력연산이 끝나기를 기다린 다음 준비대기렬을 련결한다. 한편 처리기위주의 프로쎄스는 일반적으로 실행하는동안 완료시간량자를 사용하며 즉시에 준비대기렬에로 복귀한다. 이와 같이 처리기위주의 프로쎄스들은 처리기시간의 일부를 부당하게 할당하는 경향이 있는데 그것은 입출력위주의 프로쎄스들, 입출력장치들의 비효과적인 사용 그리고 응답시간의 변동의 증가로 인하여 성능이 떨어 지는 결과를 초래한다.

[HALD 91]에서는 가상순환법(VRR)이라고 부르는 개량된 순환법을 제기하고 있는데 그것은 순환법의 부당성을 극복하고 있다. 그림 9-7에서는 그 방법을 설명해 주고 있

다. 새로운 프로쎄스들이 도착하여 준비대기렬을 련결하는데 그것은 FCFS법에 기초하여 관리된다. 실행중에 있는 프로쎄스가 시간초과되면 준비대기렬에로 복귀한다. 프로쎄스가 입출력에서 페색되면 입출력대기렬을 련결한다. 지금까지는 이것이 일반적이였다. 새로운 특징은 입출력블로크로부터 해방된후에 프로쎄스가 이동해 가는 FCFS보조대기렬이 있는 것이다. 배분결정을 하려고 할 때 보조대기렬의 프로쎄스들은 기본준비대기렬의것보다 우선적으로 선택한다. 보조대기렬로부터 마지막으로 선택된후의 실행에 소비된 총 시간을 던 시간이상은 실행되지 않는다. 성능에 대한 저자들의 연구결과는 이 방법이 개량으로 순환법을 초과한다는것을 보여 주었다.

최단프로쎄스후처리법

FCFS법에 있는 긴 프로쎄스에 유리하게 편중되는 현상을 줄이는 다른 방법이 바로 최단프로쎄스후처리(SPN)법이다. 이것은 예상되는 최단처리시간을 가지는 프로쎄스를 다음에 선택하는 비선취방책이다. 그러면 짧은 프로쎄스가 보다 긴 일감들을 넘어서 대기렬의 머리부에로 뛰여 넘는다.

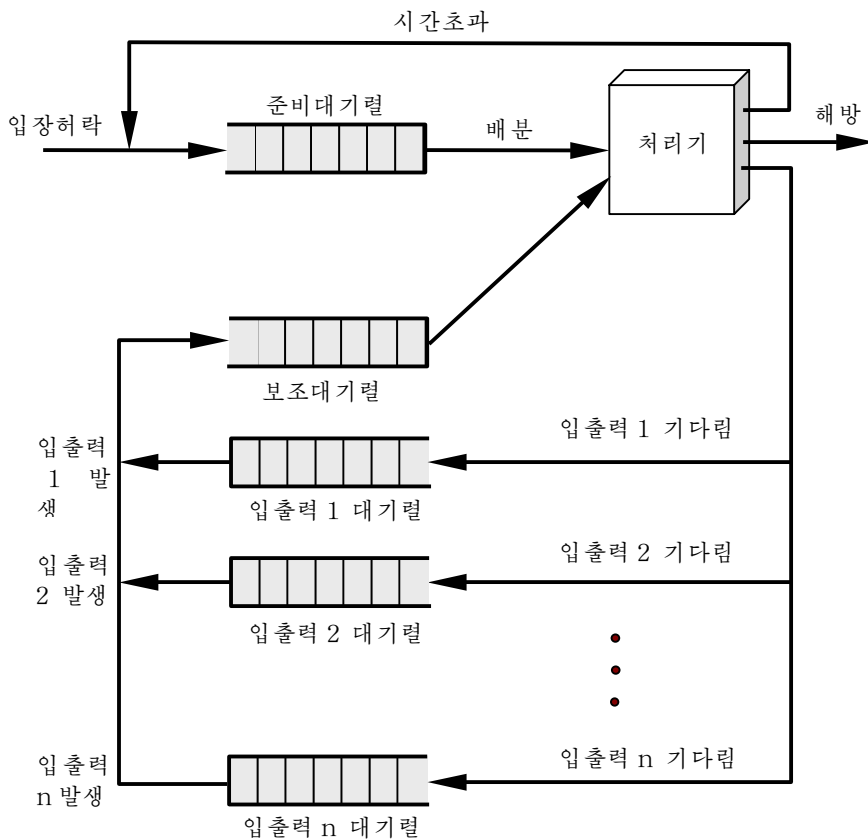



그림 9-7. 가상순환일정작성기의 대기렬도식

그림 9-5와 표 9-5는 이 실례의 결과를 보여 주고 있다. 프로쎄스 E가 FCFS에서보다 훨씬 먼저 봉사를 받는다는것에 주의하라. 또한 전반적인 성능은 응답시간에 의하여 훨씬 개선된다. 그러나 보다 긴 프로쎄스들에서 특히 응답시간의 변동이 증가되고 예측가능성이 감소된다.

SPN방책의 한가지 곤난성은 매개 프로쎄스의 요구되는 처리시간을 알거나 적어도 추

산할 필요가 있는것이다. 일괄일감들에서 체계는 프로그람작성자에게 그 값을 추산하여 조작체계에 줄것을 요구할수 있다. 만일 프로그람작성자의 추산이 충분하게 실제적인 실행시간에 들어 간다면 체계는 일감을 포기할수 있다. 생성관계에서 동일한 일감들이 자주 실행된다면 그것을 통계적으로 장악할수 있다. 대화형프로쎄스들인 경우에 조작체계는 매개 프로쎄스에 대한 매개 단락의 실행평균을 보존할수 있다. 가장 간단한 계산을 다음과 같이 할수 있다.

$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i \qquad (9\text{-}1)$$

여기서

T_i = 프로쎄스의 i번째 구체례에 대한 처리기실행시간(일괄일감의 총 실행시간; 대화형일감의 처리기단락시간)

S_i = i번째 구체례의 예측값

S_1 = 첫번째 구체례의 예측값; 계산안됨

매번 완전합을 재계산하는 작업을 피하기 위하여 이 식을 다음과 같이 다시 쓸수 있다. 즉

$$S_{n+1} = \frac{1}{n}Tn + \frac{n-1}{n}Sn \qquad (9\text{-}2)$$

이 공식에서는 매개 구체례에 동일한 무게를 주고 있다. 대표적으로는 보다 최근의 구체례에 보다 큰 무게를 주려고 한다. 왜냐하면 이것들이 앞으로의 동작을 더 잘 반영하기때문이다. 과거값들의 시간계렬에 기초하여 미래값을 예측하기 위한 일반적인 수법은 지수평균법이다. 즉

$$S_{n+1} = \alpha Tn + (1-\alpha)Sn \qquad (9\text{-}3)$$

여기서 α는 일정한 무게결수($0<\alpha<1$)로서 보다 최근의 관찰과 덜 최근의 관찰들에 주어진 상대적인 무게를 결정한다. 식 9-2과 비교하여 보라. 과거관찰수에 관계되는 상수값 α를 사용하면 모든 과거값들을 관찰하는 환경을 마련할수는 있지만 많은 구체례의것들이 더 작은 무게를 가지게 된다. 이것을 좀 더 상세히 보기 위하여 식 9-3의 확장형식을 아래에서 고찰하자. 즉

$$S_{n+1} = \alpha Tn + (1-\alpha)T_{n-1} + \ldots + (1-\alpha)^i \alpha T_{n-1} + \ldots + (1-\alpha)^n S_1 \qquad (9\text{-}4)$$

α와 $(1-\alpha)$가 1보다 작기때문에 앞의 식에서 매개 축차항은 류사하다. 실례로 α=0.8일 때 식 9-4은 다음과 같이 된다. 즉

$$S_{n+1} = 0.8T_n + 0.16T_{n-1} + 0.032T_{n-2} + 0.0064T_{n-3} + \ldots$$

관찰이 오래전의것일수록 그것은 평균값에 더 적게 반영된다.

확장시 그것의 위치의 함수로서의 결수의 크기를 그림 9-8에 제시하였다. α의 값이 클수록 보다 최근의 관찰에 주어 지는 무게는 더욱 커진다. α=0.8일 때 가상적으로 모든 무게는 네개의 가장 최근의 관찰에로 확대된다. 1에 가까운 α의 값을 사용하는것이 가지는 우점은 평균값이 급속한 변화를 즉시에 반영하는것이다. 이것의 결함은 만일 관찰량의 값에 단시간의 파동이 있고 따라서 평균값에 거꾸로 작용한다면 α의 평균값을 사용할 때 평균값에 급변하는 변화를 초래하는것이다. 그림 9-9에서는 지수평균법(α의 두개 값에서)

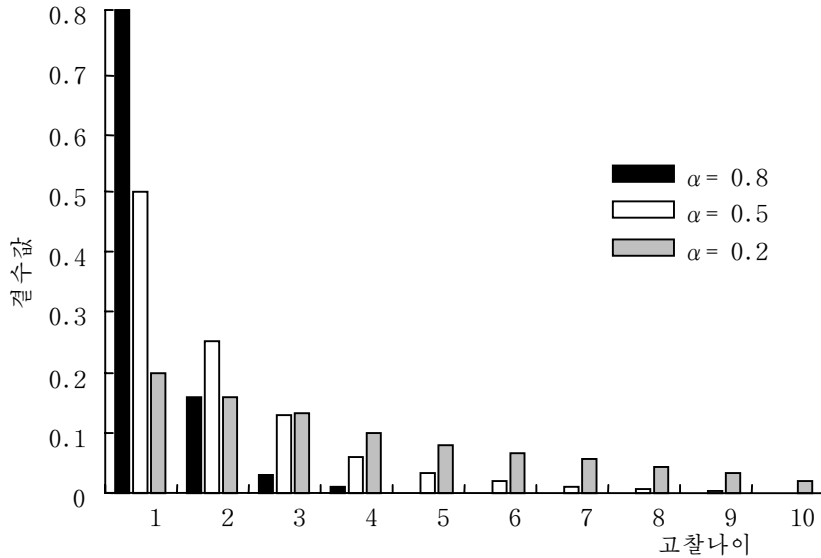



그림 9-8. 지수평활결수

과 단순평균법을 비교하고 있다. 그림 9-9 ㄱ에서는 관찰값이 1에서 시작되며 10까지 점차 증가하다가 거기에서 머무른다. 그림 9-9 ㄴ에서는 관찰값이 20에서 시작되며 10까지 점차 떨어 지다가 거기서 머무른다. 두 경우에 모두 S_1=0 이라고 추산하고 시작하였다. 이것은 새로운 프로쎄스들에 보다 큰 우선권을 준다. 지수평균법이 단순평균법보다 더 빨리 프로쎄스동작의 변화를 추적하며 α의 값이 클수록 관찰값의 변화에 보다 빨리 반응하게 된다.

한편 SPN법이 보다 긴 일감들에 유리하게 편차를 감소시킴에도 불구하고 그것은 선취가 없기때문에 아직은 시분할이나 트랜잭션처리환경에는 적합하지 않다. FCFS법에서 설명한 가장 나쁜 경우의 분석을 돌이켜 보면 프로쎄스들인 W, X, Y 그리고 Z는 짧은 과제인 Y에 강한 벌칙을 주면서 동일한 순서로 여전히 집행된다.

최단나머지시간법

최단나머지시간(SPN)법은 SPN법의 선취형이다. 이 경우에 일정작성기는 항상 예상되는 가장 짧은 나머지처리시간을 가지는 프로쎄스를 선택한다. 새로운 프로쎄스가 준비대기렬을 련결할 때 그것은 사실상 현재 실행중에 있는 프로쎄스보다 보다 짧은 나머지시간을 가질수 있다. 따라서 일정작성기는 새로운 프로쎄스가 준비될 때마다 선취할수 있다. SPN법에서와 같이 일정작성기는 선택기능을 수행하기 위하여 처리시간에 대한 추산을 해야 한다. 이때 보다 긴 프로쎄스들은 고갈의 위험성을 가진다.

SRT법에서는 FCFS법에서와 같이 긴 프로쎄스에 유리하게 편중되는 경향이 없다. 순환법에서와는 달리 보충적인 새치기가 발생되지 않으므로 간접소비시간을 줄이고 있다. 한편 경과한 봉사시간을 기록해야 하는데 이것은 간접소비시간을 증가시킨다. 또한 SRT법은 SPN법보다 더 큰 일감처리시간성능을 가진다. 그것은 짧은 일감이 보다 긴 일감보다 실행에서 즉시적인 우선권을 가지기때문이다.

실례(표 9-5)에서 가장 짧은 세개의 프로쎄스모두가 즉시 봉사를 받아 정규화된 일감처리시간이 각각 1.0으로 된다는데 주목하자.

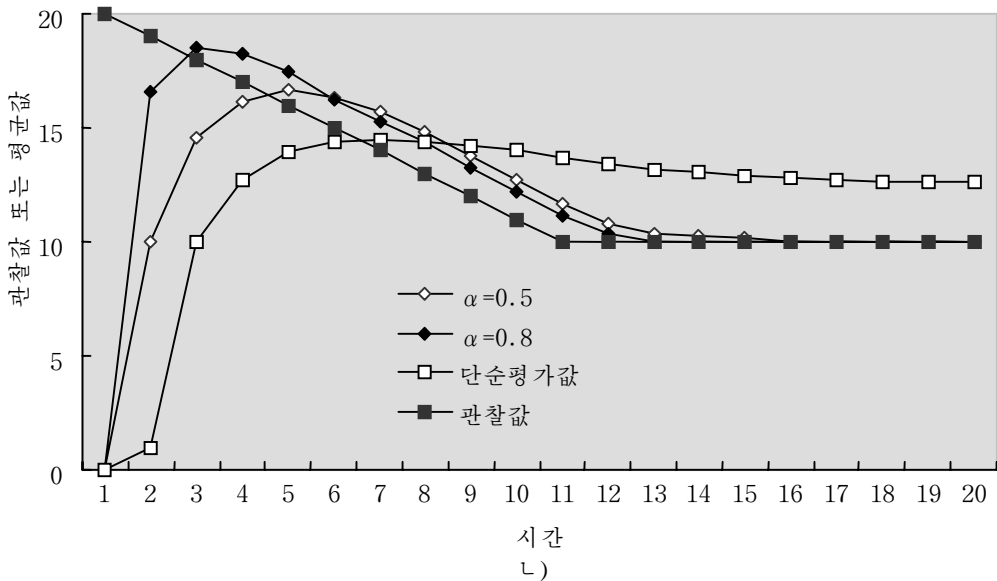
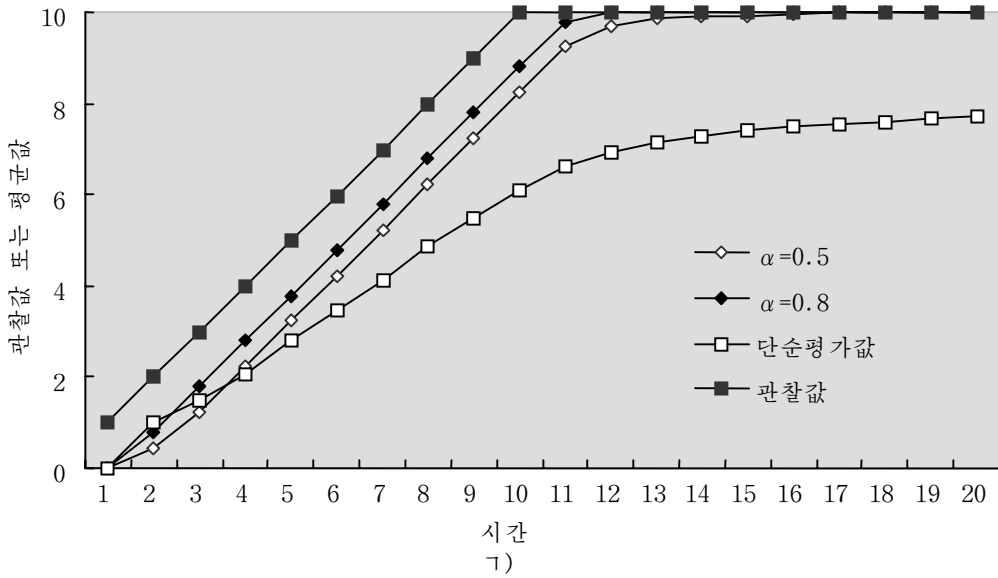



그림 9-9. 지수평균법의 사용
ㄱ-증가함수, ㄴ-감소함수

최고응답률후처리법

표 9-5에서는 지표수자로서 실제 봉사시간에 대한 일감처리시간의 비인 정규화된 일감처리시간을 사용하였다. 매개 개별적인 프로쎄스에서 이 비를 최소화하고 모든 프로쎄스에서 평균값을 최소화하는것이 필요하다. 일반적으로 봉사시간이 얼마나 되겠는가를 미리 알수는 없지만 과거경력이나 사용자나 구성관리자로부터 받는 입력에 기초하여 그것을 근사화할수는 있다. 다음의 비를 고찰하자. 즉

$$R = \frac{w + s}{s}$$

여기서 R = 응답률
W = 처리기를 기다리는데 걸린 시간
S = 예상되는 봉사시간

만일 이 값을 가지는 프로쎄스가 즉시에 배분된다면 R는 정규화된 일감처리시간과 같다. R의 최소값은 1.0인데 이것은 프로쎄스가 처음에 체계에 들어 갈 때 발생한다.

일정작성규칙은 다음과 같다. 즉 현재의 프로쎄스가 완료되거나 페색될 때 R의 제일 큰 값을 가지는 준비프로쎄스를 선택한다. 이 방법은 그것이 프로쎄스의 나이를 말해 주므로 아주 인기가 있다. 보다 짧은 일감들은 유리(분모가 커질수록 비률도 커진다.)하지만 봉사없이 나이먹음은 그 비를 증가시켜 보다 긴 프로쎄스가 결국 과거의 경쟁하는 보다 짧은 일감으로 된다.

SRT법과 SPN법에서와 같이 예상하는 봉사시간은 최고응답률후처리(HRRN)법을 사용할수 있도록 추산하여야 한다.

반결합법

만일 각이한 프로쎄스의 상대적인 길이를 지적하여 주지 않는다면 SPN법, SRT법 그리고 HRRN법중의 아무것도 사용할수 없다. 보다 짧은 일감들에 대한 우선선택권을 확립하는 다른 방도는 보다 오래동안 실행되여 온 일감들에 벌칙을 주는것이다. 다른 말로 하면 나머지실행시간에 주목할수 없다면 지금까지 실행에 걸린 시간에 주목하는것이다.

이것을 실현하는 방도는 다음과 같다. 일정작성은 선취(시간량자에서)형으로 하고 동적우선권기구를 사용한다. 프로쎄스가 처음에 체계에 들어 갈 때 그것을 RQ0에 배치한다(그림 9-4를 보시오.). 그것의 첫 실행후에 다시 준비상태로 복귀할 때 그것을 RQ1에 배치한다. 그것이 선택된후 다음번에는 그것을 우선권이 그다음으로 낮은 대기렬에로 넘긴다. 이때 보다 짧은 프로쎄스는 준비대기렬의 계층구조를 따라 멀리 이동함이 없이 급속히 완료하나 보다 긴 프로쎄스는 점차 아래쪽으로 이동한다. 그리하여 보다 새롭고 짧은 프로쎄스들이 보다 오래되고 유리하게 된다. 매개 대기렬에서 우선권이 제일 낮은 대기렬을 제외하고는 단순한 FCFS기구를 사용한다. 우선권이 제일 낮은 대기렬의 프로쎄스는 더는 낮아 질수 없지만 그것이 실행을 완료할 때까지 대기렬로 여러번 복귀된다. 그리하여 대기렬은 순환식으로 취급된다.

그림 9-10에서는 프로쎄스가 각이한 대기렬들을 뒤따르는 경로를 보여 주는것으로서 반결합일정작성기구를 설명하고 있다.[5] 이 방법은 여러주위반결합법이라고도 하는데 이것은 조작체계가 처리기를 프로쎄스에 배정하고 프로쎄스가 페색되거나 선취될 때 그것을 한개의 우선권대기렬에로 도로 보낸다는것을 의미한다.

이 방안에 대한 많은 변종들이 있다. 단순한 변종은 순환법에서와 같은 식으로 즉 주기적인 시간간격으로 선취를 진행하는것이다. 실례에서는 한개의 시간단위의 량자에서 이것(그림 9-5와 표 9-5)을 보여 준다. 이 경우에 동작은 시간량자가 1인 순환법과 류사하다는것을 주의하자.

방금 개괄한 단순방안에서 제기되는 한가지 문제는 보다 긴 프로쎄스의 일감처리시간이 급격히 늘어 날수 있다는것이다. 사실상 그것은 새로운 일감들이 자주 체계에 들어 올 때 고갈이 발생할 가능성을 준다. 이것을 보상하기 위하여 대기렬에 따라 선취시간을 변화시킬수 있다. 즉 RQ0으로부터 일정작성한 프로쎄스는 한 시간단위동안 실행할수 있게 한다음 선취하고 RQ1로부터 일정작성한 프로쎄스는두 시간동안실행할수 있게 한다.

---

5. 점선들은 그림 9-4 와 같이 가능한 이행들의 정적인 서술이 아니라 그것이 시간순서도식이라는것을 강조하기 위하여 서술하였다.

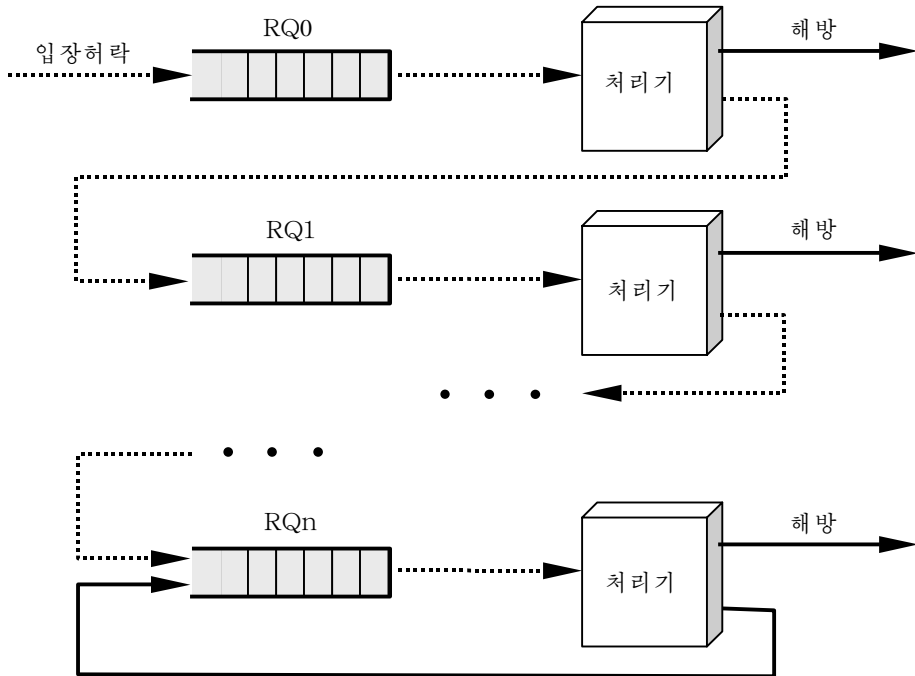



그림 9-10. 반결합일정작성법

이런 식으로 계속해 나간다. 일반적으로 RQ_i로부터 일정작성한 프로쎄스는 선취하기전에 2^i시간단위동안 실행할수 있다. 이 방안은 그림 9-5와 표 9-5의 실레에서 설명하였다.

보다 낮은 우선권으로 보다 큰 시간배정을 할수 있다. 보다 긴 프로쎄스는 여전히 고갈을 당할수 있다. 가능한 대책은 프로쎄스가 현재의 대기렬에서 봉사를 기다리면서 일정한 시간을 보낸후에 그것을 우선권이 더 높은 대기렬에로 올려 보내는것이다.

성능비교

명백히 각이한 일정작성방책들의 성능은 일정작성방책의 선택에 작용하는 결정적인 인자로 된다. 그러나 상대적인 성능이 각이한 프로쎄스들의 확률분포와 일정작성의 효률과 문맥절환기구 그리고 입출력요구의 성질과 입출력부분체계의 성능 등에 관계되므로 명확한 비교를 하는것은 불가능하다. 그렇지만 여기서는 일부 일반적인 결론을 내리기 위해 다음과 같은 해석을 한다.

대기렬짓기의 해석

이 절에서는 빠쏭도착과 지수봉사시간에 대한 일반적인 가정하에서 기본대기렬공식을 사용한다.[6]

---

6. 이 장에서 사용하는 대기렬용어는 부록 9-ㄴ에서 종합하였다. 대기렬해석에 대한 기본참고내용은 WilliamStallings.com/StudentSupport.html의 Computer Sience Student Support Site에서 찾아 볼수 있다.

먼저 봉사시간과는 관계없이 봉사하기 위하여 다음항목을 선택하는 일정작성규칙이 다음의 관계를 따른다고 하자. 즉

$$\frac{T_r}{T_s} = \frac{1}{1-\rho}$$

여기서 T_r = 일감처리시간 또는 상주시간; 기다림시간과 실행시간을 더한 체계의 총 시간

T_s = 평균봉사시간; 실행상태에서 소비한 평균시간

ρ = 처리기의 사용률

표 9-6. 두가지 우선권부류를 가진 단일봉사기대기렬에 대한 공식들

가정
1. 빠쏭도착률
2. 우선권 1항목들은 우선권 2항목들의 앞에서 봉사된다.
3. 우선권이 같은 항목들에 대해서는 선입선출배분
4. 봉사중인 때에는 어떤 항목도 새치기되지 못한다.
5. 항목들은 대기렬을 리탈하지 못한다(잃어 진 항목을 지연항목이라고 한다.).

ㄱ) 일반공식

$$\lambda = \lambda_1 + \lambda_2$$

$$\rho_1 = \lambda_1 T_{s1}; \rho_2 = \lambda_2 T_{s2}$$

$$\rho = \rho_1 + \rho_2$$

$$T_s = \frac{\lambda_1}{\lambda_2} T_{s1} + \frac{\lambda_2}{\lambda} T_{s2}$$

$$T_r = \frac{\lambda_1}{\lambda_2} T_{r1} + \frac{\lambda_2}{\lambda} T_{r2}$$

ㄴ) 새치기없음; 지수봉사시간

$$T_{r1} = T_{s1} + \frac{\rho_1 T_{s1} + \rho_2 T_2}{1-\rho_1}$$

$$T_{r2} = T_{s2} + \frac{T_{r1} + T_{s1}}{1-\rho}$$

ㄷ) 선취회복대기렬작성규칙; 지수봉사시간

$$T_{r1} = 1 + \frac{\rho_1 T_{s1}}{1-\rho_1}$$

$$T_{r2} = 1 + \frac{1}{1-\rho}\left(\rho_1 T_{s1} + \frac{\rho T_s}{1-\rho}\right)$$

특히 매개 프로쎄스의 우선권이 예상되는 봉사시간에는 무관계하게 할당되는 우선권에 기초한 일정작성기는 단순한 FCFS법의 규칙에서와 같이 동일한 평균일감처리시간과 정규화된 평균일감처리시간을 보장한다. 더우기 선취가 있는가 없는가는 이 평균에서 차이를 조성하지 않는다.

순환법과 FCFS법을 제외하고 지금까지 고찰한 여러가지 일정작성규칙들은 예상되는 봉사시간에 기초하여 선택을 한다. 공교롭게도 이러한 규칙들의 닫긴 해석모형들을 개발하는것이 매우 힘들다는것이 판명되였다. 그러나 우선권이 봉사시간에 기초를 두고 있는 우선권일정작성방법을 고찰함으로써 그러한 일정작성알고리듬들의 FCFS법에 비한 상대적인 성능에 대한 표상을 가질수 있다.

만일 일정작성이 우선권에 기초하여 진행되고 또 프로쎄스들이 봉사시간에 기초하여 우선권부류에 할당된다면 이때에는 차이점들이 명백해 진다. 표 9-6 에서는 우선권부류가 두가지이고 매개 부류에서 봉사시간이 서로 다른 경우에 해당한 공식들을 제시하였다. 표에서 λ는 도착률이다. 이 결과들은 우선권부류가 임의의 개수일 때에도 성립될수 있다. 공식들은 비선취식일정작성과 선취식일정작성인 경우에 서로 차이난다는것에 주의하자. 후자의 경우에는 우선권이 보다 높은 프로쎄스가 준비될 때 우선권이 보다 낮은 프로쎄스가 즉시에 새치기된다고 가정하고 있다.

실례로서 우선권부류가 두가지이고 매개 부류의 프로쎄스도착수가 같으며 우선권이 보다 낮은 부류의 평균봉사시간이 우선권이 보다 높은 부류에 비하여 5배인 경우를 고찰하자. 이때 보다 짧은 일감들에 우선권을 주려고 한다. 그림 9-11에서는 총체적인 결과를 보여 주고 있다. 보다 짧은 일감들에 우선권을 줌으로써 정규화된 평균일감처리시간을 개선하였다. 기대하였던바와 같이 선취권까지 사용하였을 때 일감처리시간이 제일 크게 개선되였다. 그러나 총체적인 성능은 크게 개선되지 않았다.

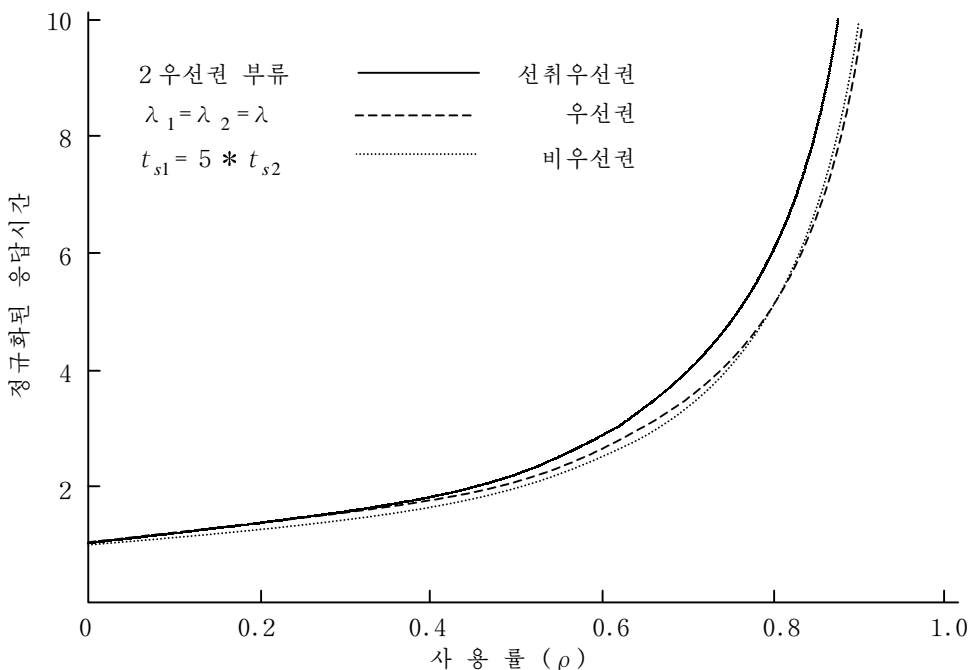



그림 9-11. 총체적으로 정규화된 응답시간

두가지 우선권부류를 각각 고찰할 때 그것들의 차이가 명백해 진다. 그림 9-12에는 우선권이 보다 높으면서도 보다 짧은 프로쎄스들에 대한 결과를 보여 주고 있다. 비교를 위해서 그라프의 맨 웃곡선은 우선권을 사용하지 않은것을 보여 주고 있는데 처리시간이 보다 짧은 모든 프로쎄스들의것보다 상대적인 성능이 절반으로 떨어 진다는것을 간단히 알수 있다. 다른 두 곡선은 보다 높은 우선권을 할당한 프로쎄스들에 대한것이다. 체계가 선취권을 가지지 않는 우선권일정작성법을 사용하여 실행하는 경우에 현저한 개선을 가져 왔다. 그것들은 선취권을 사용하였을 때 더욱 뚜렷한 개선을 보여 주었다.

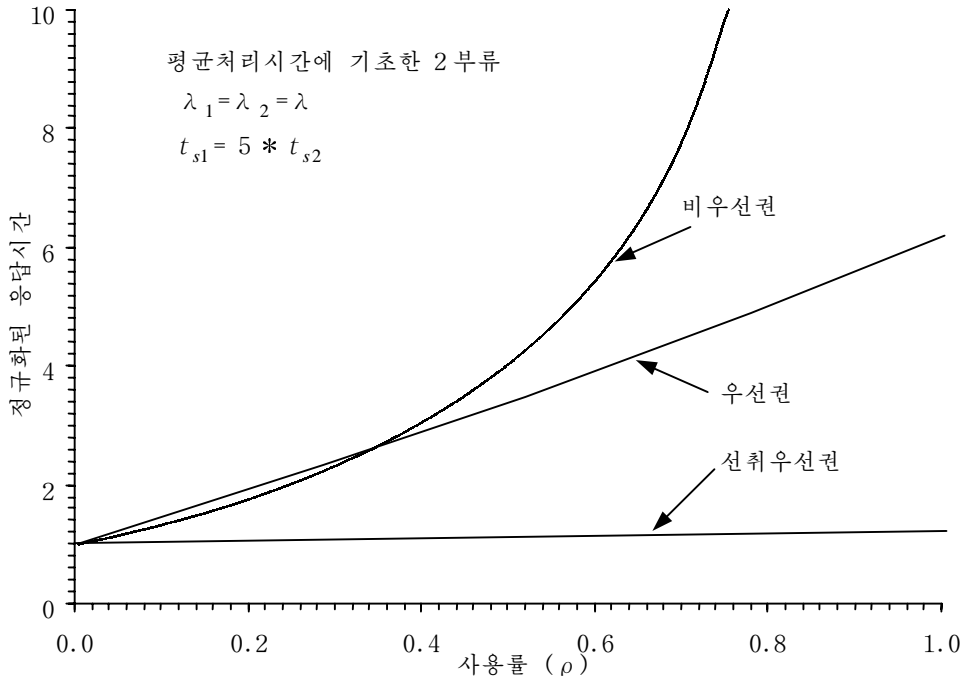



그림 9-12. 보다 짧은 프로세스들에 대한 정규화된 응답신호

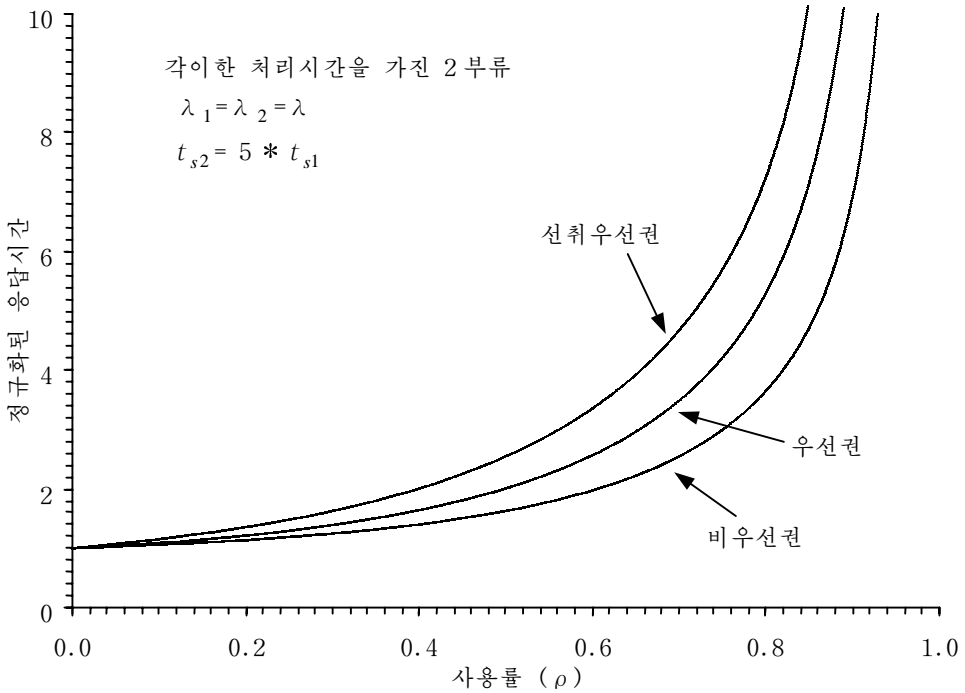



그림 9-13. 보다 긴 프로세스들에 대한 정규화된 응답시간

그림 9-13에서는 보다 우선권이 낮으면서 보다 긴 프로쎄스들에 대하여 동일한 해석을 진행한 결과를 보여 주고 있다. 예상하였던바 그대로 그러한 프로쎄스들은 우선권일정작성할 때 성능을 떨군다.

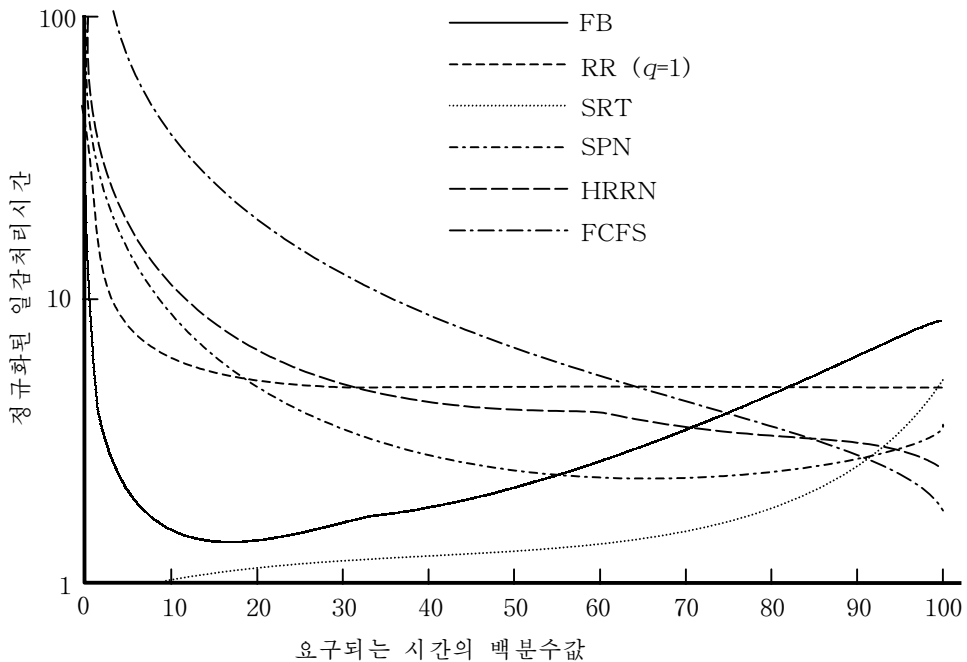



그림 9-14. 정규화된 일감처리시간에 대한 모의결과

모의모형화

해석모형화에서 제기되는 일부 난점들은 리산사건모의를 하여 극복하는데 이것은 넓은 범위의 방책들을 모형화할수 있게 한다. 모의의 결함은 주어 진 실행의 결과를 특정한 가정들을 한 조건에서 특정한 프로쎄스집단에만 적용할수 있는것이다. 그렇지만 유효한 고찰결과를 얻을수 있다.

그러한 한가지 연구결과를 [FINK 88]에서 보고하고 있다. 도착률이 λ=0.8이고 평균봉사시간이 T_S=1인 50000개의 프로쎄스에 대한 모의를 진행하였다. 이때 처리기의 사용률은 P=λT_S=0.8이라고 가정하였다. 이와 같이 한개의 사용률값이 주어 진 조건에서 측정을 하고 있다.

결과를 얻기 위하여 프로쎄스들을 봉사시간의 백분수값별로 매개가 500개씩 되게 그룹을 지었다. 이때 가장 짧은 봉사시간을 가진 500개의 프로쎄스들은 첫번째 백분수값을, 이것을 제외하고 또 가장 짧은 봉사시간을 가진 500개의 프로쎄스들은 두번째 백분수값을 가진다. 이런 식으로 그룹을 짓는다. 이렇게 하면 프로쎄스길이의 함수로서 프로쎄스들에 대한 여러가지 방책들의 효과를 조사할수 있다.

그림 9-14에서는 정규화된 일감처리시간을, 그림 9-15는 평균기다림시간을 각각 보여 주고 있다. 일감처리시간을 보면 FCFS법의 성능이 좋지 않다는것을 알수 있다. 여기서는 3분의 1에 해당하는 프로쎄스들의 일감처리시간이 봉사시간보다 10배이상으로 된다. 더우기 이것들은 가장 짧은 프로쎄스들에 해당되는것이다. 한편 일정작성이 봉사시간과는 관계없으므로 기대했던바대로 절대기다림시간은 일정하다. 그림들은 한개의 시간단위의 량

자를 사용하는 순환법을 보여 주고 있다. 한개이하의 량자에서 실행되는 가장 짧은 프로쎄스들을 제외하고 순환법은 모든 프로쎄스들에서 공정하게 약 5의 정규화된 일감처리시간을 보장한다. 가장 짧은 프로쎄스들을 제외하고 그다음에 가장 짧은 프로쎄스들은 순환법보다 더 좋게 수행된다.SPN법의 선취변종인 최첨단나머지시간법은 가장 긴 7%의 프로쎄스들을 제외하고 SPN법보다 더 좋게 수행된다. 비선취방책들가운데서 FCFS법은 긴 프로쎄스들에서, SPN은 짧은 프로쎄스들에서 특성이 더 좋다는것을 알수 있다. 다음에 가장 높은 응답률은 이 두 효과들사이의 중간에서 보장된다고 볼수 있는데 이것을 그림에서 확인할수 있다. 그림에서는 마지막으로 매개 우선권대기렬에서 고정된 균일한 량자를 가지는 반결합법을 보여 주고 있다. 기대하였던바대로 FB법은 짧은 프로쎄스들에 대하여 매우 좋은 특성을 가진다.

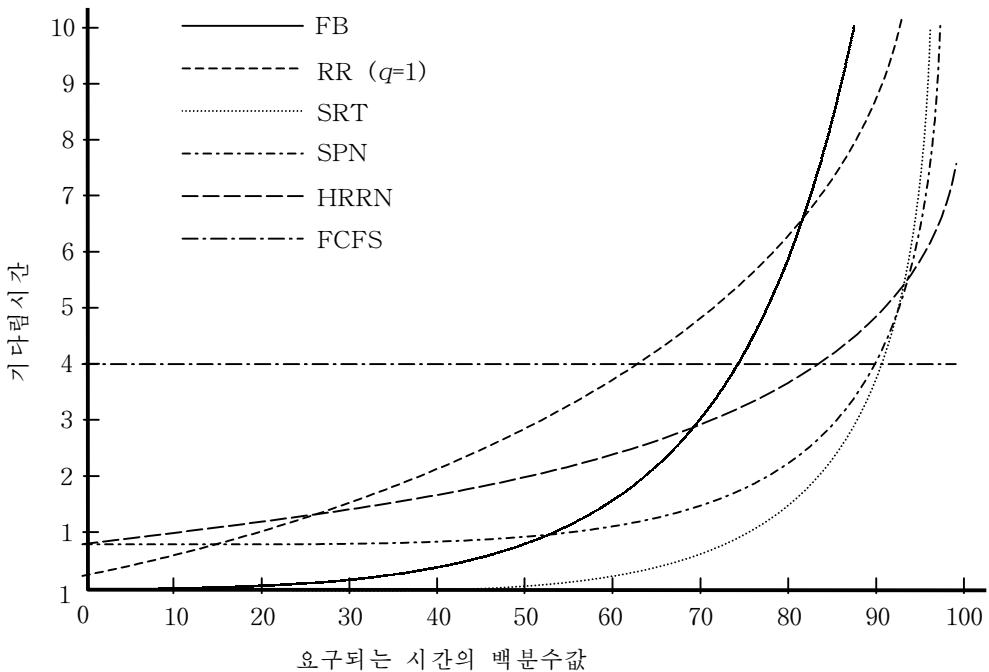



그림 9-15. 기다림시간에 대한 모의결과

공평공유일정작성법

지금까지 취급한 모든 일정작성알고리듬들은 다음에 실행하는 프로쎄스들을 선택하는 단일프로쎄스집결소로서의 준비프로쎄스집단을 취급하였다. 이 집결소는 우선권에 의하여 무너 뜨릴수는 있지만 한편으로는 같은 종류의것들이다.

그러나 다중사용자체계에서 개별사용자응용프로그람들이나 일감들을 다중프로쎄스(또는 스레드)들로서 조직할수 있는 경우에 전통적인 일정작성기로 인식되지 않는 프로쎄스집단에 대한 구성법을 적용할수 있다. 사용자의 견지에서 볼 때 제기되는 문제는 특정한 프로쎄스를 어떻게 수행하는가 하는것이 아니라 하나의 단일프로그람을 이루는 프로쎄스모임을 어떻게 수행하는가 하는것이다. 이때 프로쎄스모임에 기초하여 일정작성을 결정하는것이 중요하다. 이 방법은 일반적으로 공평공유일정작성법으로 알려 져 있다. 이 개념은 매

개 사용자가 단일프로쎄스로 표현된다고 해도 사용자가 그룹에로 확장할수 있다. 실례로 시분할체계에서 주어 진 큰 무리의 모든 사용자들을 같은 그룹의 성원이라고 볼수 있으면 좋을것이다. 이때 만일 한개의 큰 무리의 많은 성원들이 체계에 가입한다면 응답시간의 저하는 다른 큰 무리의 사용자들이 아니라 그 큰 무리의 성원들에게 주로 작용한다는것을 보게 될것이다.

용어 공평공유에는 그러한 일정작성기의 원리가 반영되여 있다. 매개 사용자에게는 체계자원들의 총체적인 사용의 일부로서 그 자원들에 대한 사용자공유를 정의하는 어떤 분류무게를 할당한다. 특히 매개 사용자에게 처리기의 공유를 할당한다. 그러한 조직은 다소 선형적으로 동작하여 만일 사용자 A가 사용자 B의 두배의 무게를 가질 때 긴 실행시에 사용자 A는 사용자 B보다 두배의 작업을 할수 있다. 공평공유일정작성기의 목적은 공평공유값보다 더 적은것을 가지는 사용자들에게 보다 많은 자원을 주는가를 감시하는것이다.

공평공유식일정작성기들에 대한 많은 안들이 제기되였다[HENR84, KAY88, WOOD86]. 이 절에서는 [HENR 84]에서 제안되고 많은 UNIX체계에서 실현된 방법을 서술한다. 그 방법을 공평공유일정작성(FSS)법이라고 부른다. FSS법은 일정작성을 결정하는데서 매개 프로쎄스의 개별적인 실행경력과 함께 관계되는 프로쎄스그룹의 실행경력을 참작한다. 체계는 사용자공동체를 공평공유그룹으로 나누고 매개 그룹에 처리기자원의 일부를 배정한다. 이때 매개를 처리기사용의 25%씩을 차지하는 네개의 그룹으로 나눌수 있다. 사실상 매개 공평공유그룹은 완전한 체계보다 상대적으로 뜨게 실행되는 가상체계를 준다.

일정작성은 우선권을 기초로 진행하는데 그것은 프로쎄스의 기본우선권, 그것의 최근 처리기사용 그리고 프로쎄스가 속한 그룹의 최근 처리기사용을 고려한다. 우선권의 수값이 클수록 우선권은 더 낮다. 그룹 k의 프로쎄스 j에 대하여 다음의 공식들을 적용할수 있다. 즉

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$GCPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i-1)}{2} + \frac{GCPU_k(i-1)}{4 \times W_k}$$

여기서

$CPU_j(i)$ = 간격 i에서 프로쎄스 j 에 의한 처리기사용률의 크기

$GCPU_k(i)$ = 간격 i에서 그룹 k의 처리기사용률의 크기

$P_j(i)$ = 간격 i의 시작점에서 프로쎄스 j의 우선권; 낮은 값일수록 우선권은 더 높다.

$Base_j$ = 프로쎄스 j의 기준우선권

W_k = 제약조건 $0 \leqq W_k \leqq 1$, $\sum_k W_k = 1$일 때 그룹 k에 할당된 무게

매개 프로쎄스에는 기준우선권이 할당된다. 프로쎄스의 우선권은 프로쎄스가 처리기를 사용할 때와 프로쎄스가 속한 그룹이 처리기를 사용할 때 인입한다. 그룹을 사용하는 경우에 평균값은 그룹의 무게로 나누어 정규화한다. 그룹에 할당된 무게가 클수록 그것의 사

시간	프로세스 A 우선권	프로세스 A 프로세스	프로세스 A 그룹	프로세스 B 우선권	프로세스 B 프로세스	프로세스 B 그룹	프로세스 C 우선권	프로세스 C 프로세스	프로세스 C 그룹
0	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0	60	0	0
1	90	30	30	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0 1 2 • • 60
2	75	15 16 17 • • 75	15 16 17 • • 75	90	30	30	75	0	30
3	96	37	37	74	15	15 16 17 • • 75	60	0 1 2 • • 60	15 16 17 • • 75
4	78	18 19 20 • • 78	18 19 20 • • 78	81	7	37	93	30	37
5	98	39	39	70	3	18	76	15	18

그룹 1 (프로세스 A) 그룹 2 (프로세스 B, C)

검은 4각형은 실행중에 있는 프로세스를 표시한다.

그림 9-16. 세개의 프로세스, 두개의 그룹을 가진 공평공유일정작성기

용은 더 적게 우선권에 영향을 준다.

그림 9-16에는 프로세스 A가 첫번째 그룹에, 프로세스 B와 C가 두번째 그룹에 있고 매개 그룹에 0.5의 무게를 주고 있는 실례를 제시하였다. 모든 프로세스들은 처리기위주형이고 일상적으로 실행준비가 되여 있다고 가정한다. 모든 프로세스들은 기준우선권 60

을 가지고 있다. 처리기사용률은 다음과 같이 측정한다. 즉 처리기는 초당 60번 새치기되고 매개 새치기시 현재 실행되고 있는 프로쎄스의 처리기사용마당은 1증가되며 대응하는 그룹의 처리기마당도 1증가된다. 초당 한번씩 우선권들은 재계산된다.

그림에서는 프로쎄스 A가 먼저 일정작성된다. 1s의 끝점에서 그것은 선취된다. 이제 프로쎄스 B 와 C는 더 높은 우선권을 가지며 프로쎄스 B가 일정작성된다. 두번째 시간단위의 끝에서 프로쎄스 A가 가장 높은 우선권을 가진다. 그러한 패턴이 되풀이된다. 즉 핵심부는 A ,B, A, C, A, B의 순서로 프로쎄스들을 일정작성한다. 이때 처리기의 50％는 첫번째 그룹에 속하는 프로쎄스 A에 배정하고 두번째 그룹에 속하는 프로쎄스 B와 C에 50％를 배정한다.

제 3 절. 전통적인 UNIX의 일정작성

이 절에서는 SVR 3과 4.3 BCD UNIX에서 사용되는 전통적인 UNIX의 일정작성을 고찰한다. 이 체계들은 주로 시분할대화형환경을 목표로 하고 있다. 일정작성알고리듬은 우선권이 낮은 배경일감이 고갈되지 않는한 대화사용자에게 좋은 응답시간을 보장하도록 설계되였다. 지금은 이 알고리듬이 현대적인 UNIX체계들에서 사용되고 있지 않지만 그것이 실제적인 실시간일정작성알고리듬들의 전형이므로 방법을 검토하는것이 유익하다. SVR 4의 일정작성방안은 실시간요구에 대한 적응성을 포함하고 있으므로 따라서 그것에 대한 설명은 제10장에서 하기로 한다.

전통적인 UNIX일정작성기는 매개 우선권대기렬에서 순환법을 사용하는 여러 준위의 반결합법을 적용하고 있다. 체계는 1s마다 선취를 한다. 즉 실행되고 있는 프로쎄스는 1s내에 페색되거나 완료되지 않는다면 그것을 선취한다. 우선권은 프로쎄스의 형태나 실행경력에 기초를 두고 있다. 다음의 공식들을 적용할수 있다. 즉

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i-1)}{2} + nice_j$$

여기서

$CPU_j(i)$ = 간격에서 프로쎄스에 위한 처리기사용률의 크기

$P_j(i)$ = 간격의 시작점에서 프로쎄스의 우선권; 낮은 값일수록 우선권이 더 높다.

$Base_j$ = 프로쎄스의 기준우선권

$nice_j$ = 사용자조종이 가능한 조절결수

매개 프로쎄스의 우선권은 초당 한번씩 재계산되는데 이때 새로운 일정작성결정이 진행된다. 기준우선권의 목적은 모든 프로쎄스들을 고정된 대역의 우선권준위들로 분할하는것이다. CPU와 nice의 요소들을 프로쎄스가 그것이 할당된 대역(기준우선권준위에 의하여 할당된)밖으로 이동하는것을 예방할수 있게 제한을 받는다. 이 대역들은 블로크장치(즉 디스크)에 대한 접근을 최적화하고 조작체계가 체계호출에 즉시 응답할수 있게 하는데 사용한다. 우선권이 감소하는 순서로 보면 대역들은 다음과 같다.

- 디스크블로크교체

- 블로크입출력장치의 조종
- 파일조작
- 글자입출력장치의 조종
- 사용자프로세스

시간	프로세스 A 우선권	프로세스 A CPU 계수	프로세스 B 우선권	프로세스 B CPU 계수	프로세스 C 우선권	프로세스 C CPU 계수
0	60	0 1 2 • • 60	60	0	60	0
1	75	30	60	0 1 2 • • 60	60	0 1 2 • • 60
2	67	15	75	30	60	0 1 2 • • 60
3	63	7 8 9 • • 67	67	15	75	30
4	76	33	63	7 8 9 • • 67	67	15
5	68	16	76	33	63	7

검은 4각형은 실행중에 있는 프로세스를 표시한다.

그림 9-17. 전통적인 UNIX의 프로세스일정작성의 실례

이 계층구조는 입출력장치들을 가장 효과적으로 사용할수 있도록 한다. 사용자프로쎄스대역안에서 실행경력을 사용하는 경우에 입출력위주의 프로쎄스들의 희생으로 처리기위

주의 프로쎄스들을 벌칙하는 경향이 있다. 이것은 특히 효률을 개선한다. 순환선취방안을 맞물리면 일정작성전략은 일반목적의 시간분할을 위한 요구사항들을 만족시킬수 있게 갖추어 진다.

그림 9-17에는 프로쎄스일정작성의 실례를 제시하였다. 프로쎄스 A, B 그리고 C는 기준우선권 60을 가지고 동시에 창조된다(세밀한 값은 무시한다.). 박자에 대하여 초당 60번체계가 새치기되며 실행되고 있는 프로쎄스에 대하여 계수기를 1증가시킨다. 이 실례에서는 어느 프로쎄스도 자기들을 폐색시키지 않으면 다른 프로쎄스들은 실행준비가 되여 있지 않다고 가정하였다. 이것을 그림 9-16과 비교하여 볼것을 권고한다.

요약, 기본용어 및 복습문제

조작체계는 프로쎄스들의 실행에서 세가지 형태의 일정작성을 결정해야 한다. 장기일정작성은 새로운 프로쎄스들이 체계에 입장허락될 때 결정된다. 중기일정작성은 교체기능의 일부로서 프로그람이 주기억기에 부분적으로 또는 완전히 옮겨 져 실행될수 있을 때 결정된다. 단기일정작성은 준비프로쎄스가 처리기에 의하여 다음에 실행되게 될 때 결정된다. 이 장에서는 단기일정작성과 관련된 문제들을 집중적으로 고찰하였다.

단기일정작성설계에서는 여러가지 기준이 사용되였다. 이 기준들중에서 일부는 개별적인 사용자들에 의해 인식되는 체계의 동작과 관련(사용자지향)되고 다른것들은 모든 사용자들의 요구를 만족시키는데서 체계의 총체적인 값과 관련되며 다른 기준들은 실제상 보다 정성적인 성질을 가진다. 사용자의 견지에서 볼 때 가장 중요한 특성지표는 일반적으로 응답시간이고 체계의 견지에서 볼 때 중요한것은 처리능력 또는 처리기의 사용률이다.

준비된 모든 프로쎄스들사이에서 단기일정작성결정을 하기 위한 여러가지 알고리듬들이 개발되였다. 즉

- **선래선봉사법 :** 봉사를 위해 가장 오래동안 기다리고 있는 프로쎄스를 선택한다.
- **순환법 :** 임의의 실행중에 있는 프로쎄스를 처리기시간의 짧은 구간에로 제한하도록 시간세분법을 사용한다. 이것을 준비된 모든 프로쎄스들사이에서 순회한다.
- **최단프로쎄스후처리법 :** 예상되는 최단프로쎄스시간으로 프로쎄스를 선택한다. 이때 프로쎄스는 선취하지 않는다.
- **최단나머지시간법 :** 예상되는 최단나머지시간으로 프로쎄스를 선택한다. 프로쎄스는 다른 프로쎄스가 준비될 때 선취될수 있다.
- **최고응답률후처리법 :** 정규화된 일감처리시간을 추산한데 기초하여 일정작성결정을 한다.
- **반결합법 :** 일정작성대기렬모임을 만들고 프로쎄스를 실행경력과 다른기준에 맞는 대기렬에 배정한다.

일정작성알고리듬의 선택은 예상되는 성능과 실현의 복잡성에 관계된다.

기본용어

도착률 배분기 공평공유식일정작성법 공평성 선래선봉사(FCFS)법 선입선출(FIFO)법	장기일정작성기 중기일정작성기 예측 상주시간 응답시간 일정작성우선권	봉사시간 단기일정작성기 처리능력 시간세분법 일감처리시간(TAT) 사용률 기다림시간

복습문제

1. 처리기일정작성의 세가지 형태에 대하여 간단히 설명하시오.
2. 대화형조작체계에서 보통 한계성능요구라는것은 무엇인가?
3. 일감처리시간과 응답시간의 차이는 무엇인가?
4. 프로쎄스일정작성에서 낮은 우선권값이 낮은 우선권을 나타내는가 아니면 높은 우선권을 나타내는가?
5. 선취일정 및 비선취일정작성의 차이는 무엇인가?
6. FCFS일정작성법을 간단히 정의하시오.
7. 순환일정작성법을 간단히 정의하시오.
8. 최단프로쎄스후처리일정작성법을 간단히 정의하시오.
9. 최단나머지시간일정작성법을 간단히 정의하시오.
10. 최고응답률후처리일정작성법을 간단히 정의하시오.
11. 반결합일정작성법을 간단히 정의하시오.

참 고 문 헌

조작체계들과 관련된 모든 책들에서는 실제적으로 일정작성방법들을 취급하고 있다. 각이한 일정작성방책들에 대한 엄밀한 대기렬해석은 [STUC 85], [KLEI 76] 그리고 [CONW 67]에서 진행하고 있다. [DOWD 93]에서는 여러가지 일정작성알고리듬들에 대한 교육학적인 성능분석을 하고 있다.

CONW67 Conway, R.; Maxwell, W.;and Miller, L. *Theory of Scheduling.* Reading, MA:Addison-Wesly, 1967.

DOWD93 Dowdy, L., and Lowery, *C.P.S.to Operating Systems.* Upper Saddle River, NJ:Prentice Hall, 1993.

KLEI76 Kleinrock, L. *Queuing Systems, Volume II:Computer Applications.* New York: Wiley, 1976.

STUC85 Stuck, B., and Arthurs, E. *A computer and Communications Network Performance Analysis Primer.* Englewood Cliffs, NJ:Prentice Hall, 1985.

련 습 문 제

1. 다음의 프로쎄스모임을 보자.

프로쎄스이름	도착시간	처리시간
A	0	3
B	1	5
C	3	2
D	9	5
E	12	5

이 모임에 대하여 표 9-5와 그림 9-5에서 서술한것과 같은 분석을 하여 보시오.

2. 다음의 프로쎄스모임에 대하여 문제 9-1을 되풀이하시오.

프로쎄스이름	도착시간	처리시간
A	0	1
B	1	9
C	2	1
D	3	9

3. 비선취일정작성알고리듬들가운데서 SPN법이 동시에 도착하는 일감묶음에 대하여 최소평균기다림시간을 보장한다는것을 증명하시오.

4. 프로쎄스에 단락시간패턴이 6, 4, 6, 4, 13, 13, 13과 같고 초기추정이 10이라고 하자. 이때 그림 9-9의것과 같은 곡선을 그리시오.

5. 식 9-3대신에 다음의 한 쌍의 등식을 고찰하자. 즉

$$S_{n+1} = \alpha Tn+(1-\alpha Sn)$$
$$X_{n+1} = \min[Ubound,\ \max[Lbound,\ (\beta Sn+1)]]$$

여기서 *Ubound* 와 *Lbound* 는 추산된 *T* 값의 우 및 아래경계들을 미리 취한다. X_{n+1}의 값은 최단프로쎄스후처리알고리듬에서 S_{n+1} 대신에 사용한다. α와 β는 어떤 기능들을 수행하는가? 그리고 그것들의 더 높은 값들의 효과는 무엇인가?

6. 비선취단일처리기체계에서 준비대기렬은 어떤 일감의 완료후 즉시에 시간 t에서 세개의 일감을 포함한다. 시간 t_1, t_2및 t_3에서 세개의 일감들은 추산된 실행시간 r_1, r_2 및 r_3을 각각 가진다. 그림 9-18에서는 시간에 따르는 응답률들이 직선적으로 증가하는것을 보여 주고 있다. 이 실례를 사용하여 최소최대응답률일정작성법이라고 부르는 응답률일정작성의 방안을 세워 보시오. 최소최대응답률일정작성은 그이상의 도착들은 무시하고 주어 진 일감묶음에 대한 최대응답률을 최소화한다(요령: 어떤 일감을 마지막의것으로 일정작성하겠는가를 먼저 결심하시오.).

7. 앞의 문제에서 최소최대응답률알고리듬에 주어 진 일감묶음에 대하여 최대응답률을 최소화한다는것을 증명하시오(요령: 최고응답률을 달성하게 될 일감과 그전에 실행된 모든 일감들에 주목하시오. 임의의 다른 순서로 일정작성된 동일한 일감부분모임을 고려하시오. 그리고 그것들가운데서 마지막것으로 실행되는 일감의 응답률에 주목하시오. 이 부분모임이 총체적인 모임의 다른 일감들과 혼합될수 있다는것에 주의하시오.).

8. 프로쎄스가 기다리고 봉사 받는데 소비한 평균 총 시간으로서 상주시간 T_r를 정의하시오. 평균봉사시간이 T_S인 FCFS법에서 $T_r = T_S / (1-\rho)$임을 증명하시오. 여기서 ρ는 사용률이다.

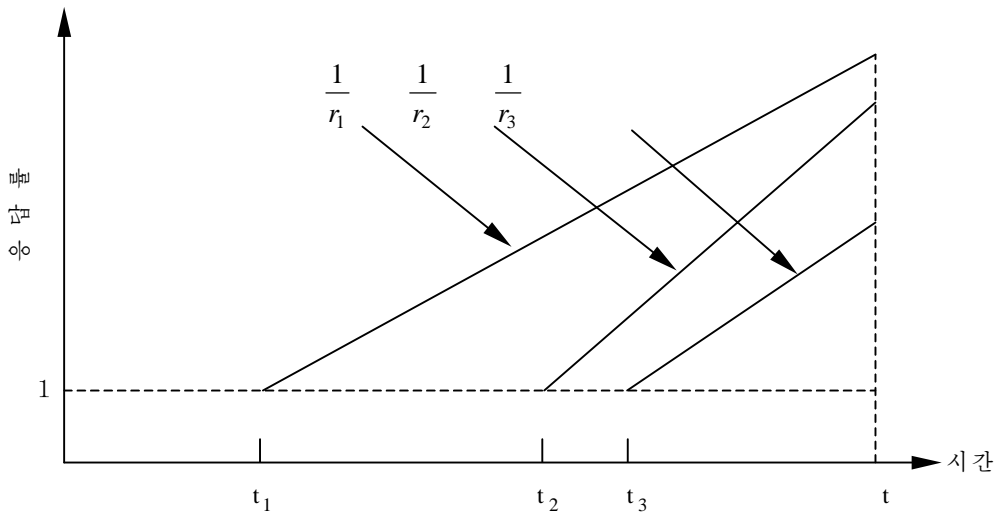



그림 9-18. 시간의 함수로서의 응답률

9. 어떤 처리기가 간접소비시간이 없는 준비된 대기렬에 있는 모든 프로쎄스들속에서 다중화되고 있다(평균봉사시간보다 대단히 작은 시간구간을 사용하는 준비된 프로쎄스들에서 리상화한 순환일정작성모형이다.). 지수봉사시간을 가진 무한원천으로부터 들어 오는 빠쏭입력에서 봉사시간이 x인 프로쎄스의 평균응답시간 R_x가 $R_x = x/(1-\rho)$로 주어 진다는것을 증명하시오(요령: william Stallings.com / Student Support.html의 Queuing Analysis document 에서 기본대기렬방정식을 상기하고 주어 진 프로쎄스의 도착에 기초하여 체계에서 기다리는 항목수 w를 고려하시오.).

10. 대부분의 순환일정작성기는 고정된 크기의 량자를 사용한다. 먼저 작은 량자에 유리하게 변수를 주시오. 그리고 큰 량자에 유리하게 변수를 주시오. 변수들을 적용한 체계와 일감들의 형태를 비교하시오.

11. 대기렬체계에서 새로운 일감들은 봉사 받기전에 얼마동안 기다려야 한다. 일감이 기다리는동안 그것의 우선권은 시간에 따라 령으로부터 α의 속도로 직선적으로 증가한다. 일감은 그것의 우선권이 봉사 받는 일감의 우선권에 이를 때까지 기다린다. 그다음 순환법을 사용하여 봉사에서 다른 일감과 똑같이 처리기를 공유하기 시작한다. 그리고 그것의 우선권은 보다 뜬 속도 β로서 계속 증가한다. 이 알고리듬을 자기본위순환법이라고 부른다. 그것은 봉사 받는 일감이 자기의 우선권을 계속 증가시킴으로써 처리기를 독점하려고 하기(무익하게)때문이다. 그림 9-19를 사용하여 봉사시간이 x인 일감에서 평균응답시간 R_x가 다음의 식으로 표시된다는것을 증명하시오. 즉

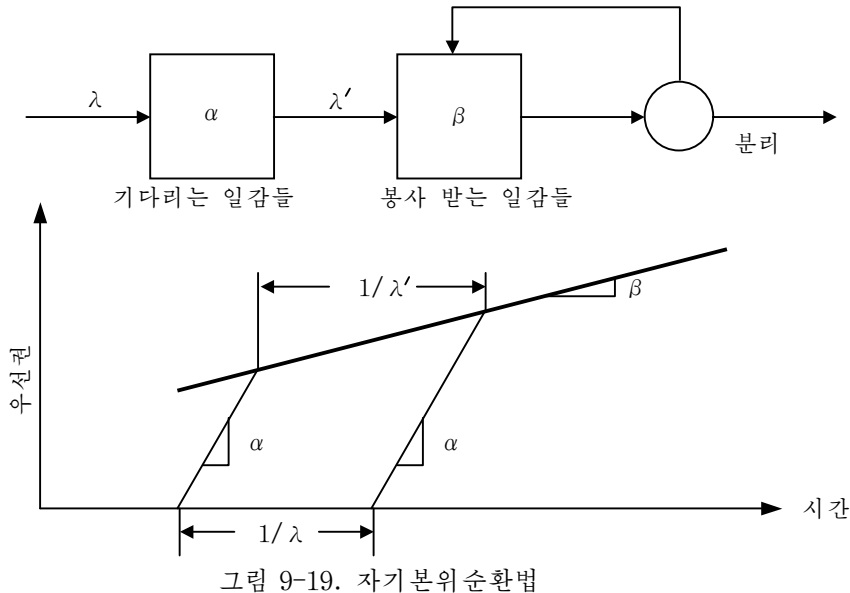
$$R_x = \frac{s}{1-\rho} + \frac{x-s}{1-\rho'}$$

여기서 $\rho = \lambda s \qquad \rho' = \rho(1 - \frac{\beta}{\alpha}) \qquad 0 \le \beta < \alpha$

도착과 봉사시간이 각각 $1/\lambda$과 S로서 지수함수적으로 분포되여 있다고 가정한다(요령: 총체적인 체계와 두개의 부분체계를 따로따로 고찰하시오.).

12. 순환일정작성법을 사용하는 대화형체계는 보통의 요청들에 다음과 같이 담보된 응답을 주려고 한다. 즉 모든 준비된 프로쎄스들사이에서 순환주기를 끝낸후에 체계는 최대응답시간을 봉사를 요구하는 프로쎄스수로 나눔으로써 다음주기동안 매개 준비된 프로쎄스에 배정할 시간구간을 결정한다. 이것이 합리적인 방책인가?

13. 일반적으로 어떤 형태의 프로쎄스가 여러준위반결합대기렬일정작성기에 유리한가? 말하자면 처리기위주의 프로쎄스인가 아니면 입출력위주의 프로쎄스인가? 왜 그런지를 간단히 설명하시오.

그림 9-19. 자기본위순환법

14. 우선권에 기초한 프로쎄스일정작성에서 일정작성기는 현재 준비상태에서 보다 높은 우선권을 가진 다른 프로쎄스가 없을 때 특정한 프로쎄스에로 조종을 넘긴다. 프로쎄스일정작성을 결정하는데 다른 정보를 사용하지 않는다고 가정한다. 또한 프로쎄스우선권들이 프로쎄스창조시간에 설정되여 변화되지 않는다고 가정한다. 이러한 가정하에 동작하는 체계에서 Dekker의 해(제5장 제2절을 보라.)를 호상배제문제에 사용하는것이 왜 《위험》한가? 이 질문에 대한 대답으로 바라지 않는 사건이 발생할수 있고 또 그것이 어떻게 발생할수 있는가를 설명하시오.

15. A～E의 다섯개의 묶음일감이 콤퓨터에 본질상 동시에 도착한다. 그것들에 대하여 추산된 실행시간이 각각 15, 9, 3, 6, 12min이다. 그것들의 우선권(외부적으로 정의됨)들은 각각 6, 3, 7, 9, 4인데 낮은 값일수록 우선권은 더 높다. 다음의 매개 일정작성알고리듬들에서 매개 프로쎄스에 대한 일감처리시간과 모든 일감들에 대한 평균일감처리시간을 결정하시오. 프로쎄스를 절환하는데 드는 간접소비시간은 무시한다. 마지막 세가지 경우에 오직 한개의 일감만이 그것이 끝날 때까지 한번에 실행되고 모든 일감들은 완전히 처리기위주형이라고 가정한다.

ㄱ) 시간량자가 1min인 순환법
ㄴ) 우선권일정작성법
ㄷ) FCFS(15, 9, 3, 6, 12의 순서로 실행)법
ㄹ) 최단일감우선처리법

부록 9-ㄱ. 응답시간

응답시간은 주어 진 입력에 체계가 반작용하는데 드는 시간이다. 대화형트랜잭션에서 그것은 사용자가 마지막건반을 쳐서부터 콤퓨터에서 얻어 진 결과가 화면에 표시되기 시작한 순간까지의 시간으로서 정의할수 있다. 각이한 형태의 응용에서 정의가 조금씩 다를수 있다. 일반적으로 응답시간은 체계가 특정한 과제를 수행하기 위한 요청에 응답하는데 드는 시간이다. 리상적으로는 임의의 응용에서 응답시간이 짧은것이 좋다. 그러나 거의 모든 경우에 응답시간이 짧을수록 거기에 드는 가격은 더 높아 진다. 가격은 다음의 두가지에 원천을 두고 있다. 즉

- **콤퓨터처리능력 :** 콤퓨터의 속도가 높을수록 응답시간은 더 작다. 물론 처리능력이 증가되면 가격도 증가한다.
- **경쟁하는 요구 :** 어떤 프로쎄스에 빠른 응답시간을 보장하면 다른 프로쎄스는 그만큼 곤난해 진다.

이때 응답시간에 제시된 준위의 값은 응답시간을 달성하기 위한 가격과 반대의 관계를 가진다.

표 9-7. 응답시간의 범위

15s이상 이것은 회화형상호작용에 배치된다. 일부 형태의 응용에서 일부 사용자들은 하나의 단순한 문의에 대한 대답을 받기 위하여 15s이상 말단앞에 앉아 있는것을 만족해 할수 있다. 만일 그러나 바쁜 사람이 15s 이상이나 잡혀 있는것은 허용할수 없다. 만일 그러한 지연이 발생하면 사용자가 다른 동작을 하다가 얼마 지나서 응답을 요청할수 있도록 체계를 설계하여야 한다.
4s이상 이것은 운영자가 근거리기억기(콤퓨터의 기억기가 아니라 운영자의 기억기)에 정보를 의뢰하도록 요청하는 회화에서는 일반적으로 너무 길다. 그러한 지연은 문제를 푸는 작업에서는 극히 금지되고 자료입력작업을 헛되이 할수 있다. 그러나 작업을 기본상 끝낸후에는 4～15s의 지연을 허용할수 있다.
2～4s 2s이상의 지연은 높은 집중을 요구하는 말단조작에서는 금지될수 있다. 말단에서 2～4s의 기다림은 부차적인 종결이 된후에 이 범위의 지연을 허용할수 있다.
2s이하 말단사용자가 여러개의 응답에 걸쳐 정보를 기억해야 하는 경우에 응답시간은 짧아야 한다. 보다 상세한 정보를 기억시킬수록 응답에 대한 요구는 더욱 높아 져 2s는 중요한 응답시간한계로 된다
1s이하 사고력이 집중되는 작업 특히 도형처리응용에서 오랜 시간동안 사용자의 흥미와 주의를 끌기 위하여서는 응답시간이 대단히 짧아야 한다.
수분의 1s 건반치기와 화면에 표기된 글자의 보기 또는 마우스로 화면의 바꾸기에 대한 응답은 거의 동시적인 응답다시말하여 작업후 0.1s이하로 되여야 한다. 마우스와의 상호작용은 설계자가 외국어문장(지령, 뜻기호, 구도점찍기 등)을 사용하지 않는다면 극히 고속일것을 요구한다.

응답시간을 여섯가지의 일반적인 범위로 분류한 표 9-7은 [MART 88]을 참고한것이다. 1s이하의 응답시간을 요구할 때에는 설계상 곤난한 문제에 부닥치게 된다. 1s이하의 응답시간에 대한 요구는 조립흐름선과 같이 련속진행되는 외부작업을 조종하거나 특수한 방법으로 대화를 진행하는 체계에서 발생된다. 여기서 요구사항은 간단하다. 자료입력응용과 같은 사람-콤퓨터대화에서는 회화에 필요한 응답시간을 보장해야 한다. 이런 경우에는 짧은 응답시간에 대한 요구가 제기되지만 그만한 시간을 보장하는것은 어려울수 있다.

대화형응용들에서 생산성을 높이는 열쇠가 바로 빠른 응답시간에 있다는것이 많은 론문들에서 확인되였다[SHNE 84, THAD 81, GUYN 88]. 이 론문들에서는 콤퓨터와 사용자가 대방이 기다리지 않아도 되는 속도로 대화를 한다면 생산성이 현저히 제고되며 콤퓨터로 수행되는 작업의 가격과 같이 개선될 경향이 있다는것을 보여 주고 있다. 사람이 다음에 수행할 과제를 생각해야 하므로 대부분의 대화형응용들에서 2s까지의 상대적으로 뜬 응답시간은 허용되였다. 그러나 지금은 빠른 응답시간을 보장할수 있기때문에 생산성이 증가되는것을 볼수 있다.

응답시간에 대하여 보고된 결과들은 직결트랜잭션들의 분석에 기초한것이다. 트랜잭션은 말단에서 오는 사용자의 지령과 체계의 대답으로 이루어 진다. 이것은 직결체계의 사용자들을 위한 기본작업단위이다. 이것은 두개의 시간순차렬로 나눌수 있다. 즉

- 사용자응답시간 : 사용자가 지령에 대한 완전한 대답을 받은 순간과 다음지령에 들어 가는 순간사이의 시간간격
- 체계응답시간 : 사용자가 지령에 들어 가는 순간과 완전한 응답이 말단에 표시되는 순간사이의 시간간격구간

체계응답시간의 축소효과를 보여 주는 실례로서 그림 9-20에서는 집적회로소편과 기판설계를 위한 콤퓨터지원설계도형프로그람을 사용하여 기술자가 수행한 연구결과를 보여 주고 있다[SMIT 83]. 매개 트랜잭션은 어떤 방법으로 선택하는 기술자에 의한 지령과 화면에 표시하는 도형화상으로 되여 있다. 결과는 체계응답시간이 떨어 질 때 트랜잭션속도가 증가되다가 일단 체계응답시간이 1s아래로 떨어 지면 시간이 떨어 질 때 사용자응답시간도 떨어 지는것이다. 이것은 근거리기억기나 사람의 주의집중시간의 효과로 보아야 한다.

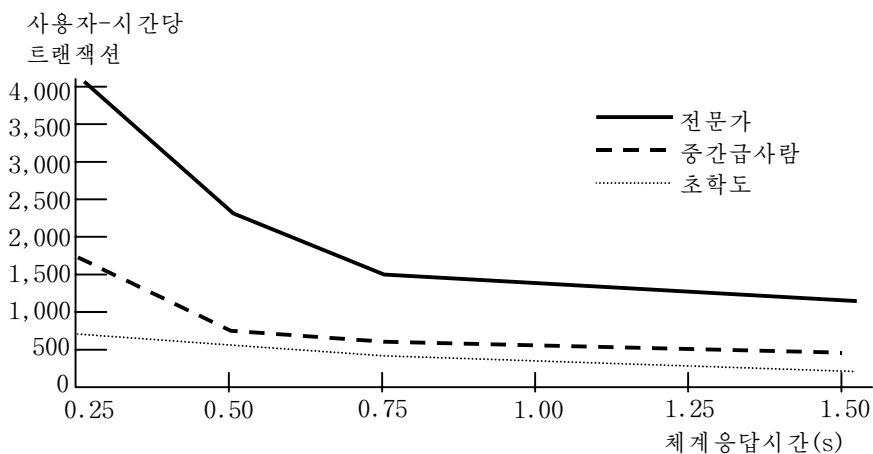



그림 9-20. 고기능도형처리에 대한 응답시간결과

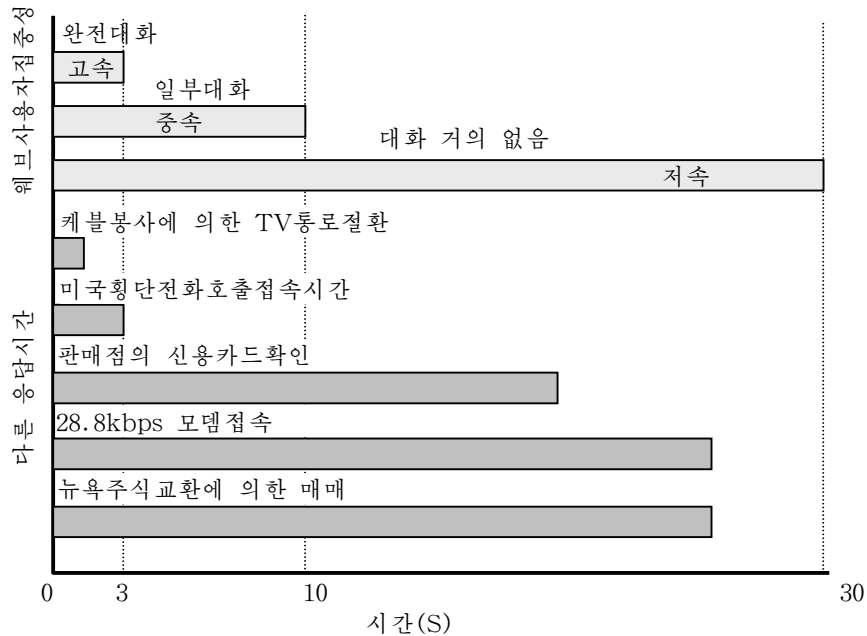



그림 9-21. 응답시간의 요구사항[SEVC 96]

응답시간이 중요하게 제기되는 다른 령역은 인터네트나 공동의 인트라네트를 넘어 선 광지역망의 사용령역이다. 전형적인 Web페지가 사용자의 화면에 나타나도록 하는데 걸리는 시간은 크게 변화된다. 응답시간은 대화조종시에 사용자관련준위에 기초하여 측정할 수 있다. 특히 매우 빠른 응답시간을 보장하는 체계는 많은 사용자들의 주의를 끈다. 그림 9-21에서와 같이 3s나 그보다 좋은 응답시간을 보장하는 Web체계들은 높은 준위의 사용자집중성을 가진다. 3~10s의 응답시간인 경우에는 일부 사용자집중성을 잃게 되며 10s 이상의 응답시간인 경우에는 사용자가 대화조종을 포기할수 있다.

부록 9-ㄴ. 대기렬체계

이 장과 다음의 몇개 장에서는 대기렬리론의 결과들을 사용한다. 이 부록에서는 대기렬체계에 대한 간단한 정의를 하고 기본용어들을 설명한다. 대기렬해석과 친숙하지 못한 독자들이 참고할 내용은 William Stallings com/Student Support.html의 Computer Science Student Support Site에서 볼수 있다.

단일봉사기대기렬

가장 단순한 대기렬체계를 그림 9-22에 제기하였다. 체계의 중심요소는 봉사기인데 이것은 항목들에 대한 봉사를 준다. 어떤 항목모집단의 항목들이 봉사받으려고 체계에 도착한다. 만일 봉사기가 비여 있으면 항목은 즉시에 봉사 받는다. 아니면 도착하는 항

목은 기다림줄에 들어 선다.[7] 봉사기가 항목의 봉사를 완료하면 그 항목은 탈퇴한다. 만일 대기렬에서 기다리는 항목들이 있다면 한개의 항목이 즉시에 봉사기에로 배분된다. 이 모형에서 봉사기는 항목집단에 대한 어떤 기능이나 봉사를 수행하는 임의의것을 표시한다(즉 처리기가 프로쎄스에 봉사하는것, 전송선로가 자료의 파케트나 프레임에 봉사하는것, 입출력장치가 입출력요청들에 대한 읽기나 쓰기봉사를 진행하는것).

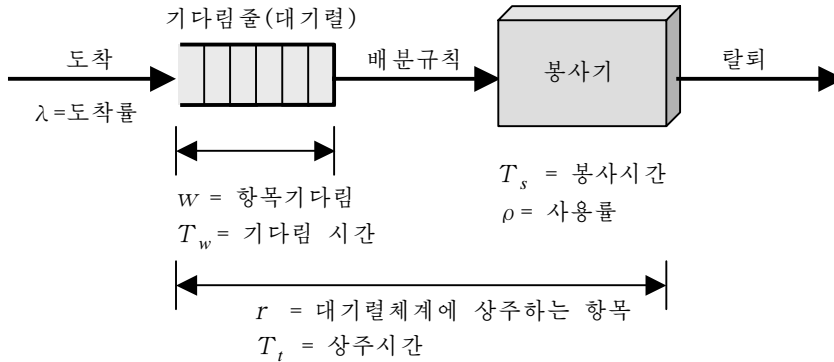



그림 9-22. 단일봉사기대기렬을 위한 대기렬작성체계의 구조와 파라메터

표 9-8에서는 대기렬모형과 관련된 몇가지 중요한 파라메터들을 요약하였다. 항목들은 평균속도(초당 도착하는 항목수) λ로 설비에 도착한다. 주어 진 순간에 어떤 개수의 항목들이 대기렬에서 기다린다(행 또는 그이상의 항목). 평균기다림수는 w이고 항목이 기다려야 하는 평균시간은 T_W이다. T_W는 전혀 기다리지 않는것까지 포함하여 들어 오는 모든 항목들에 대한 평균값이다. 봉사기는 평균봉사시간 T_S로 들어 오는 항목들을 처리한다. 이것은 봉사기에 대한 항목의 배분과 항목의 봉사기로부터 분리사이의 시간간격이다. 사용률 ρ는 어떤 시간간격에서 봉사기가 차지하는 시간의 몫이다. 마지막으로 두개의 파라메터를 전체로서의 체계에 적용한다. 즉 그것은 봉사 받는 항목(만일 있다면)들과 기다리는 항목(만일 있다면)들을 포함하여 체계에 상주하는 평균항목수 r와 항목이 기다리는 시간과 봉사 받는 시간을 포함하여 체계에서 소비한 평균시간 T_r이다. T_r를 평균상주시간이라고 한다.[8]

만일 대기렬의 용량이 무한하다고 가정하면 체계로부터 빠져 나가는 항목은 없다. 그것들은 봉사 받을수 있을 때까지 지연될뿐이다. 이러한 환경에서 탈퇴률은 도착속도와 같다. 체계를 통과하는 흐름속도인 도착속도가 증가할 때 사용률은 증가되고 그것과 함께 밀집도도 증가한다. 이때 대기렬은 더 길어 지고 기다림시간도 길어 진다. ρ=1일 때 봉사기는 포화되여 100%의 시간을 작업한다. 이때 체계에 의하여 처리할수 있는 리론적인 최대입력속도는 다음과 같다. 즉

$$\lambda_{\max} = \frac{1}{T_s}$$

그러나 ρ=1일 때 대기렬은 한계가 없이 늘어 나 체계포화에 가깝게 대단히 커진다.

표 9-8. 대기렬체계의 표기법

---

7. 일부 참고문헌에서는 기다림줄을 대기렬이라고도 부르는데 옹근체계를 대기렬이라고 부르는것이 일반적이다. 다른 지적이 없는한 기다림줄을 표현하는데서 대기렬이라는 용어를 사용하기로 한다.

8. 일부 문헌들에서는 이것을 평균대기시간이라고 하고 한편 다른데서는 대기렬에서 기다리는데 보낸(봉사 받기전에) 평균시간을 표현하는데 평균대기시간을 사용한다.

λ=도착속도; 초당 평균도착속도
T_s=매개 도착에 대한 평균봉사시간; 대기렬에서 기다리는 시간은 셈에 넣지 말고 봉사 받고 있는 시간량
ρ =사용률설비(봉사기 또는 봉사기들)가 차지하는 시간몫
w=봉사 받기를 기다리는 평균항목수
T_w=평균기다림시간(기다려야 하는 항목들과 기다림시간이 0인 항목들을 포함하여)
r=체계에 상주하는 평균항목수(기다리고 있거나 봉사 받는)
T_r=평균상주시간; 항목이 체계에서 소비하는 시간(기다리고 있거나 봉사 받는)

응답시간에 대한 요구, 완충기의 크기 등 실제적인 리유로 하여 단일봉사기의 입력속도는 리론적최대값의 70～90%범위로 제한한다.

대표적으로 다음의 가정들을 한다. 즉

- **항목모집단 :** 대표적으로 무한모집단을 생각한다. 이것은 도착속도가 모집단의 류실에 의하여 변하지 않는다는것을 의미한다. 만일 모집단이 유한이라면 도착에 쓸수 있는 모집단은 현재 체계에 있는 항목수에 따라서 감소된다. 이것은 전형적으로 도착속도를 비례적으로 감소시킨다.
- **대기렬크기 :** 대표적으로 무한한 대기렬의 크기를 생각하자. 이때 기다림줄은 제한없이 길어 진다. 유한대기렬에서는 항목들이 체계에서 류실될수 있다. 실제상 임의의 대기렬은 유한이다. 많은 경우에 이것은 해석에서 본질적인 차이가 없다.
- **배분규칙:** 봉사기가 자유상태에 있을 때 그리고 한개이상의 항목이 기다리고 있을 때 결정은 다음에 배분해야 할 항목에 대하여 하여야 한다. 제일 간단한 방법은 선입선출법이다. 이 규칙은 용어 대기렬이 언제 사용되는가를 정상적으로 암시하여 준다. 다른 방법은 후입선출법이다. 실천에서 만나게 되는것은 봉사시간에 기초한 배분규칙이다. 실례로 파케트절환형매듭은 최대우선법(가장 빨리 나가는 파케트를 발생시킬수 있는)이나 최장우선법(전송시간에 비하여 처리시간을 최소화할수 있는)에 기초하여 파케트들을 배분하도록 선정할수 있다. 유감스럽게도 봉사시간에 기초한 규칙은 해석적으로 모형화하기가 매우 어렵다.

다중봉사기대기렬

그림 9-23에는 다중봉사기를 가지고 모두가 공통대기렬을 공유하는 일반화된 단순한 모형을 제시하였다. 만일 어떤 항목이 도착하고 적어도 한개의 봉사기가 사용가능하면 그 항목은 즉시 봉사기에로 배분된다. 여기서 모든 봉사기는 동등하다고 가정한다. 이때 만일 한개이상의 봉사기가 사용가능하다고 하면 항목에 대하여 어느 봉사기를 선택할것인가를 구별할 필요가 없다. 만일 모든 봉사기들이 차지되여 있다면 대기렬이 형성되기 시작한다. 그러다가 한개의 봉사기가 자유상태로 되자마자 한개의 항목이 배분규칙에 따라 강제로 대기렬에서 빠져 나와 봉사기에로 배분된다.

그림 9-22에 제시한 모든 파라메터들중에서 사용률만 제외하고는 동일한 상호작용하에 다중봉사기의 경우에로 확장할수 있다. N개의 동등한 봉사기가 있다면 ρ는 매개 봉사기의 사용률이고 체계전체의 사용률은 $N\rho$라고 볼수 있다. $N\rho$를 보통 흐름강도 u라고 한다. 그러면 리론적인 최대사용률은 $N\times100\%$이고 리론적인 최대입력속도는 다음과 같다.

$$\lambda_{\max} = \frac{N}{T_s}$$

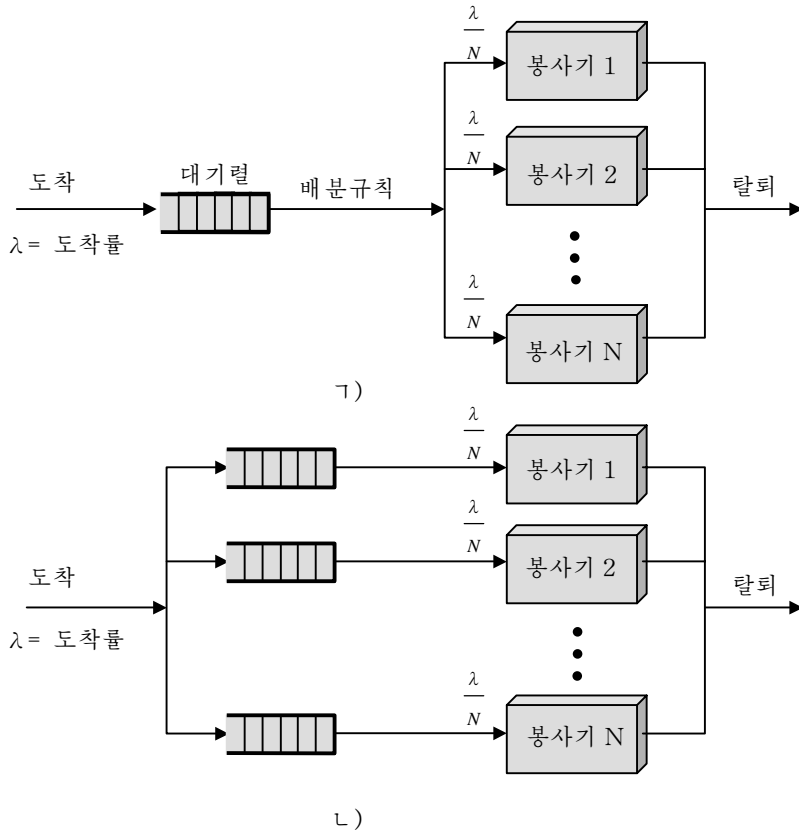



그림 9-23. 다중봉사기대 다중단일봉사기대기렬
ㄱ-다중봉사기대기렬, ㄴ-다중단일봉사기대기렬

다중봉사기대기렬의 대표적인 특성지표들은 단일봉사기대기렬의것에 대응한다. 즉 모든 봉사기들이 단일무한대기렬을 공유하는 무한모집단과 무한대기렬크기를 가정한다. 달리 지적하지 않는이상 배분규칙으로서는 FIFO법을 사용한다. 다중봉사기인 경우에 모든 봉사기들이 동등하다면 기다리는 항목을 위한 특정한 봉사기의 선택은 봉사시간에 영향을 주지 않는다.

비교를 위해 그림 9-23 ㄴ에는 다중단일봉사기대기렬의 구조를 제시하였다.

제 10 장. 다중처리기와 실시간일정작성

이 장에서는 프로쎄스의 일정작성에 대한 고찰을 계속한다. 먼저 한개이상의 처리기 사용에 의해 발생되는 문제들을 검토한다. 여기에는 많은 문제들이 있다. 다음은 다중처리체계에서 프로쎄스일정작성을 고찰한다. 또한 다중처리기의 스레드일정작성에서 제기되는 서로다른 몇가지 실제 문제들을 조사한다. 이 장의 제 2 절에서는 실시간일정작성을 취급한다. 먼저 실시간프로쎄스들의 특징지표들을 설명한다. 일정작성프로쎄스의 성질에 대하여 언급한다. 실시간일정작성의 두가지 방법 즉 교착일정작성과 속도공평일정작성에 대하여 검토한다.

제 1 절. 다중처리기의 일정작성

콤퓨터체계가 한개이상의 처리기를 내장하고 있을 때에는 일정작성기능을 설계하는데서 몇가지 새로운 문제가 제기된다. 먼저 다중처리기들에 대하여 간단히 개괄하고 다음 프로쎄스준위와 스레드준위에서 일정작성을 할 때 고려해야 할 몇가지 문제를 고찰한다.

다중처리기체계는 다음과 같이 분류할수 있다. 즉

- **성긴결합형, 분산형다중처리기체계 또는 클라스터:** 매개 처리기는 자체의 주기억기와 입출력통로들을 가지고 있으면서 상대적으로 서로 독립인 체계들로 이루어져 있다. 이러한 형태의 구성에 대해서는 제 13 장에서 해설한다.
- **기능전용처리기체계:** 실례로 입출력처리기를 들수 있다. 이 경우에는 범용처리기인 주처리기가 있다. 전용처리기들과 관련된 문제들은 제 11 장에서 해설한다.
- **밀결합형다중처리체계:** 공동의 주기억기를 공유하면서 조작체계의 통합적인 조종하에 동작하는 처리기모임으로 이루어 진다.

이 절에서는 마지막 부류에 대하여 관심을 돌리면서 특히 일정작성과 관련되는 문제들을 취급한다.

표 10-1. 동기화립도 및 프로쎄스

립도크기	설 명	동기화의 간격 (명령수)
미세립도	단일명령흐름에 고유한 병렬기구	< 20
중간립도	단일한 응용에서 병렬처리 또는 다중과제작성	20 ~ 200
립도가 큰	다중프로그람작성환경에서 동시프로쎄스의 다중처리	200 ~ 2000
대단히 립도가 큰	단일계산환경을 형성하기 위한 망마디들에 걸치는 분산처리	2000 ~ 1M
독립	다중무관계프로쎄스	(N/A)

립도

다중처리기들을 특징짓고 다른 구성방식들과 관련시켜 평가하는 좋은 방도는 체계안의 프로쎄스들사이에서 동기화의 립도나 동기화의 주파수를 비교하여 보는것이다.

알갱이의 등급에 따라 서로 다른 병렬기구를 다섯가지 부류로 분류할수 있다.이것은 [GEHR87]과 [WOOD89]에서 인용하여 표 10-1에 요약하였다.

독립병렬기구

독립병렬기구에서는 프로쎄스들사이에 명백한 동기화가 존재하지 않는다. 매개는 분리된 독립적인 응용이거나 일감이다. 이러한 형태의 병렬기구의 대표적인 사용실례로서 시분할체계를 들수 있다. 매개 사용자는 문서처리나 표계산자료의 사용과 같은 특정한 응용프로그람을 수행한다. 다중처리기는 다중프로그람작성방식의 단일처리기와 같은 봉사를 준다. 한개이상의 처리기를 사용할수 있기때문에 사용자들에 대한 평균응답시간은 보다 작아 진다.

매개 사용자에게 개인용콤퓨터나 워크스테이션을 보장함으로써 류사한 성능개선을 가져 올수 있다. 만일 임의의 파일이나 정보를 공유하려고 한다면 개별적인 체계들은 망에 의하여 받쳐 주는 분산체계에 다같이 련결되여야 한다. 이 방법에 대하여서는 제 13장에서 고찰한다. 한편 단일한 다중처리기들의 공유체계는 많은 경우에 디스크들과 다른 주변장치들의 규모가 절약되므로 분산체계보다 가격상 훨씬 효과적이다.

립도가 큰 및 대단히 큰 병렬기구

립도가 큰 및 대단히 큰 병렬기구에서는 매우 큰 준위에서 프로쎄스사이의 동기화가 진행된다. 이러한 경우에는 프로쎄스들이 다중프로그람작성방식의 단일처리기에서 실행되는 동시프로쎄스들의 모임과 같이 쉽게 처리되며 사용자쏘프트웨어를 거의 변화시킴이 없이 다중처리기에 줄수 있다.

다중처리기를 사용하는 응용실례가 [WOOD89]에 제시되여 있다. 저자들은 쏘프트웨어의 어떤 부분을 재구축하기 위하여 재콤파일해야 하는 파일들의 명세들을 작성하고 이 콤파일들중의 어느것들(보통은 모든것)을 동시에 실행시킬수 있는가를 결정하는 프로그람을 개발하였다. 저자들은 다중처리기의 속도가 디스크캐쉬(제 11장에서 한개의 제목으로 취급한다.)에서의 협동동작과 기억기에 일단 한번만 적재하면 되는 콤파일러코드의 공유로 하여 사용중에 있는 처리기의 수만큼 단순히 합계하면 되므로 예상하였던것보다 실제상 더 제고된다고 보고하고 있다.

일반적으로 통신하거나 동기화할 필요가 있는 동시프로쎄스들의 집단은 다중처리기 구성방식을 사용하는것이 유익하다. 프로쎄스들사이에서 매우 드물게 대화를 진행하는 경우에는 분산체계가 좋은 결과를 줄수 있다. 그러나 만일 대화작용이 어느정도 자주 진행될 때에는 망을 걸치는 간접소비시간에 의하여 잠재적으로 속도를 높일수 없다. 그러한 경우에는 다중처리기조직이 가장 효과적인 대책으로 된다.

중간립도병렬기구

제 4 장에서는 한개의 응용프로그람을 한개의 프로쎄스에 포함되여 있는 스레드들의 집단으로서 효과적으로 실현할수 있다는것을 보았다. 이 경우에 응용프로그람의 병렬화의 가능성은 프로그람작성자에 의하여 명확히 명시되여야 한다. 대표적으로 응용프로그람의 스레드들사이에서는 비교적 높은 등급의 동작일치와 대화를 요구한다. 이것은 중간

알갱이급의 동기화에로 이끌어 간다.

한편 독립적인, 대단히 굵은 그리고 립도가 큰 병렬화는 다중프로그람작성방식의 단일처리기나 일정작성기능에 거의 영향을 주지 않는 다중처리기에 적용할수 있는데 스레드의 일정작성시에는 그것을 재검토할 필요가 있다. 응용프로그람에서 각이한 프로쎄스들이 빈번히 대화를 하므로 한개의 스레드와 관련된 일정작성결정이 응용프로그람전체의 성능에 영향을 줄수 있다. 이 문제에 대하여서는 이 절의 마지막 부분에서 보기로 한다.

미세립도병렬기구

미세립도병렬기구는 스레드들의 사용에서 본것보다 훨씬 더 복잡한 병렬기구를 사용한다. 병렬도가 높은 응용프로그람들에 대한 연구가 많이 진행되였지만 아직까지는 전용화되고 단편적인 분야로 되고 있다. 여러가지 방법들을 적용한 좋은 참고문헌으로서는 [ALMA89]가 있다.

설계문제

다중처리기에 대한 일정작성에는 서로 련관된 문제가 있다. 즉

- 처리기들에 대한 프로쎄스들의 할당
- 개별적인 처리기들에 다중프로그람작성법의 사용
- 프로쎄스들의 실효적인 배분

이 세가지 문제를 대하는데서 적용한 방법이 일반적으로 응용프로그람의 알갱이의 등급과 사용가능한 처리기수에 관계된다는것을 명심하는것이 중요하다.

처리기들에 프로쎄스들의 할당

다중처리기의 구성방식이 처리기가 주기억기와 입출력장치들의 접근과 관련하여 특정한 물리적인 우점을 가지고 있지 않다는 점에서 균일형이라고 가정할 때 가장 단순한 일정작성방법은 처리기들을 공동자원으로 취급하고 프로쎄스들을 요구에 따라 처리기에 할당하는것이다. 여기서 문제로 제기되는것은 할당이 정적인가 또는 동적인가 하는것이다.

만일 프로쎄스가 활성화되여서부터 완료될 때까지 한개의 처리기에 계속 할당된다면 매개 처리기에 대하여 독점된 단기대기렬이 유지된다. 이 방법의 우점은 처리기의 할당이 한번만 진행되면 되기때문에 일정작성작업에서 간접소비시간이 적은것이다. 처리기를 독점적으로 사용하여 일정작성을 하는 전략을 후에 다시 취급하겠지만 그룹식 또는 무리식 일정작성이라고 하고 있다.

정적할당의 결함은 한쪽의 처리기가 만가동하는 동안 빈 대기렬을 가진 다른쪽의 처리기가 놀수 있는것이다. 이러한 현상을 예방하기 위하여 공동대기렬을 사용한다. 이때에는 모든 프로쎄스들이 한개의 총적인 대기렬에 들어 섰다가 사용가능한 임의의 처리기에로 일정작성된다. 이때 프로쎄스의 수명이 다 될 때까지 그것은 각이한 시간에 각이한 처리기에서 실행될수 있다. 밀결합형공유기억기방식에서는 모든 처리기들에 대한 상태정보를 모든 처리기들에서 사용할수 있고 따라서 프로쎄스를 일정작성하는 비용은 프로쎄스가 일정작성되는 처리기의 상태에 무관계하다.

프로쎄스들이 처리기들에 독점되는가 되지 않는가에는 관계없이 프로쎄스들을 처리기들에 할당하는 방법이 필요하다.여기에는 두가지 방법 즉 주/종법과 동위법이 사용되고 있다. 주/종방식에서 조작체계의 기본핵심기능은 항상 특정한 처리기에서 실행된다.

다른 처리기들은 다만 사용자프로그람들만을 집행할수 있다. 주처리기는 일감들의 일정작성을 할 책임을 지고 있다. 일단 프로쎄스가 능동으로 되고 종속처리기가 봉사(실례로 입출력호출)를 요구한다면 그것은 주처리기에 요청을 보내야 하며 봉사기가 실행되기를 기다려야 한다. 이 방법은 매우 단순한 방법으로서 단일처리기의 다중프로그람방식의 조작체계를 조금 강화할것을 요구한다. 한개의 처리기가 모든 기억기와 입출력자원들의 조종을 진행하므로 충돌문제는 쉽게 해결된다. 이 방법에는 두가지 결함이 있다. 즉 (1) 주처리기의 고장이 체계전체를 정지시키는것이고 (2) 주처리기가 실행에서 병목상태에 빠질수 있는것이다.

동위방식에서 조작체계는 임의의 처리기에서 실행할수 있고 매개 처리기는 사용가능한 프로쎄스들의 집결소로부터 독립적으로 일정작성을 한다. 이 방법은 조작체계를 복잡하게 한다. 조작체계는 두개의 처리기가 한개의 프로쎄스를 선택하지 않도록 하며 프로쎄스들이 대기렬에서 어떻게 해서든지 소실되지 않도록 담보해야 한다. 자원들에 대한 경쟁적인 요구들을 해결하고 동기화를 위한 수법들을 적용해야 한다.

물론 이 두 극단사이의 문제를 해결하는 방법들이 있다. 방법은 공정한 처리대신에 몇개의 처리기들을 핵심부처리에 독점시키는것이다. 다른 방법은 핵심부프로쎄스들과 다른 프로쎄스들의 요구들사이의 차이를 우선권과 집행경력에 기초하여 단순히 관리하는것이다.

개별적인 처리기들에서 다중프로그람작성법의 사용

매개 프로쎄스가 자기의 수명동안 처리기에 정적으로 할당될 때 새로운 질문이 제기된다. 즉 처리기가 다중프로그람작성방식의것인가 하는것이다. 여기서 문제는 이 질문이 왜 제기되는가 하는것이다. 프로쎄스가 입출력을 기다리거나 병행성동기화의 고려로 인하여 자주 폐색되는 경우에 처리기를 한개 프로쎄스에 얽어 매 놓는것은 특별히 랑비인것처럼 보인다.

립도가 큰 또는 독립적인 동기화의 립도를 취급(표 10-1을 보시오.)하는 전통적인 다중처리에서 사용률을 높이고 따라서 성능을 더 좋게 하기 위하여 매개 개별적인 처리기를 많은 프로쎄스들사이에서 절환할수 있게 하는것이 좋을것이다. 그러나 많은 처리기들을 가진 다중처리기에서 실행되는 중간립도의 응용프로그람인 경우에는 사정이 그리 명백하지 않다. 많은 처리기들이 사용가능할 때 매개 독립적인 처리기가 가능한 많이 차지될것이라는것은 그리 중요하지 않다. 그보다 관심을 끄는것은 응용프로그람들에 대하여 평균적으로 가장 좋은 성능을 보장하는것이다. 많은 스레드들로 이루어 지는 응용프로그람은 모든 스레드들이 동시에 실행될수 없다면 불충분하게 실행될수 있다.

프로쎄스의 배분

다중처리기의 일정작성과 관련된 마지막 설계문제는 실행해야 할 프로쎄스를 효과적으로 선택하는것이다. 다중프로그람방식의 단일처리기에서 우선권이나 과거의 사용정도에 기초한 정교한 일정작성알고리듬들을 적용함으로써 단순한 선래선보관법이상으로 성능을 개선할수 있다는것은 이미 지적하였다. 그러나 다중처리기들을 고찰할 때에는 이러한 복잡성이 필요 없거나 지어는 비생산적일수 있다. 보다 단순한 방법의 간접소비시간이 더 적으므로 보다 효과적일수 있다. 스레드일정작성의 경우에는 우선권이나 실행경력보다 더 중요한 새로운 문제들이 제기된다. 이러한 문제들을 차례로 해설한다.

프로쎄스일정작성

대부분의 전통적인 다중처리기체계들에서는 프로쎄스들을 처리기들에 독점시키지 않는다. 모든 처리기들을 위한 한개의 대기렬이 있기는 하지만 어떤 종류의 우선권방안을 사용한다면 우선권에 기초한 다중대기렬을 두고 모두가 공동집결소의 처리기들에 공급한다. 어떤 경우라도 체계를 다중봉사기의 대기렬방식이라고 볼수 있다.

2중처리기체계에서 매개 처리기의 속도는 단일처리기체계의 처리기의 처리속도의 절반으로 된다. [SAUE81]에서는 FCFS 일정작성법을 순환법과 최단나머지시간법과 비교하는 대기렬분석을 하고 있다. 여기서는 처리기가 요구하는 처리기시간으로 측정되는 프로쎄스봉사시간, 총체적인 일감시간이나 프로쎄스가 처리기를 사용할 준비가 될 때마다 요구하는 시간과 관련되는 연구를 진행하였다. 순환법의 경우에는 시간량자가 상대절환의 간접소비시간보다는 크고 평균소비시간보다는 작다고 가정하고 있다. 결과들은 봉사시간들에 보게 되는 가변성에 중요하게 관계된다. 가변성에 대한 일반적인 측정지표는 가변결수 C_S이다.[1] $C_S=0$의 값은 가변성이 없는 경우에 해당된다. 그것은 모든 프로쎄스들의 봉사시간이 같은 경우이다. C_S의 값이 증가된다는것은 봉사시간들사이에서 가변성이 증가하는것을 의미한다. C_S의 값이 5 또는 그이상으로 되는것은 처리기봉사시간의 분산에서 그리 이상한것이 아니다.

그림 10-1 ㄱ에서는 C_S의 함수로서 순환법의 처리능력을 FCFS 법의 처리능력과 비교하고 있다. 일정작성알고리듬의 차이는 2중처리기의 경우에 훨씬 적다는것을 지적해 둔다. 두개 처리기인 경우 FCFS법에서는 봉사시간이 긴 프로쎄스가 훨씬 적다. 이때 다른 프로쎄스들은 다른 처리기를 사용할수 있다. 류사한 결과들은 그림 10-1 ㄴ에 보여 주고 있다.

[SAUE81]에서는 다중프로그람작성의 등급, 입출력위주의 프로쎄스와 CPU 위주의 프로쎄스들의 혼합, 우선권의 사용 등에 대한 여러가지 가정을 한 조건에서 우와 같은 분석을 되풀이하고 있다. 일반적으로 결론할수 있는것은 한개의 처리기를 가진 경우에 비하여 두개의 처리기를 가진 경우에 특정한 일정작성규칙은 그다지 중요하지 않다는것이다. 이러한 결론은 처리기의 수가 증가할 때 더욱 확고하다. 간단한 FCFS 규칙이나 정적우선권방안내에서 FCFS의 사용은 다중처리기체계에서 적합할수 있다.

스레드일정작성

이미 설명한바와 같이 스레드에 관해서는 실행에 대한 개념이 프로쎄스에 대한 정의에서와 차이난다. 응용프로그람은 동일한 주소공간에서 협동하면서 병행하여 실행되는 스레드들의 모임으로 실행될수 있다. 단일처리기에서는 스레드들을 프로그람을 구조화하는 도구로 사용할수도 있고 입출력을 처리기와 겹치게 하는데도 사용할수 있다. 이러한 환경에서 스레드들은 응용프로그람에서 알맞춤한 병렬화를 실현하는데 쓸수 있다. 만일 응용프로그람의 각이한 스레드들이 개별적인 처리기들에서 동시에 실행된다면 극적인 성능향상을 얻을수 있다. 그러나 스레드들사이에 중요한 상호작용(중간립도병렬화)을 요구하는 응용프로그람들에서는 스레드관리와 일정작성에서의 자그마한 차이들이 성능에 중요한 영향을 줄수 있다는것을 알아야 한다[ANDE89].

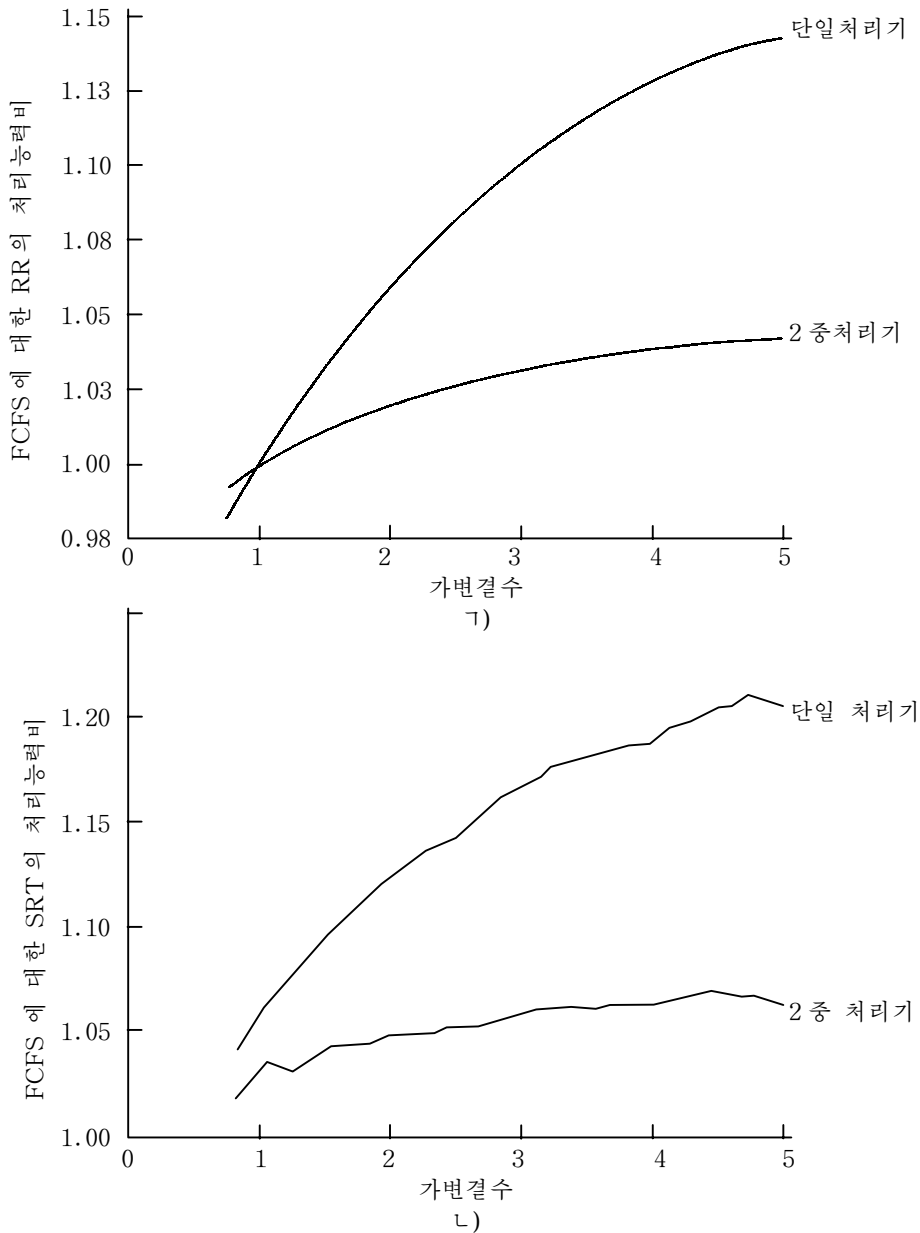



그림 10-1. 단일 및 2중 처리기에 대한 일정작성성능의 비교

다중처리기들에서의 스레드일정작성에 대한 연구가 활발히 진행되고 있는데 여기서는 기본문제와 방법들에 대하여 개괄한다. 즉

- **부하공유법:** 프로쎄스는 특정한 처리기에 할당되지 않는다. 준비프로쎄스들의 전역대기렬이 유지되고 매개 처리기는 비여 있을 때 대기렬부터 스레드를 선택한다. 부하공유라는 용어는 이 전략을 보다 영구적인 토대에 의하여 작업이 배정되는 부하균형잡기방안과 구별하기 위하여 사용한다

([CFEIT90a]를 보시오.).[2]

- **무리일정작성법:** 관련되는 스레드들의 모임은 일대일의 기준에 기초하여 동일한 시간에 처리기모임에서 실행할수 있도록 일정작성을 한다.
- **전용처리기할당법:** 이것은 부하공유법의 반대로서 처리기들에 대한 스레드들의 할당에 의하여 정의된 암시적인 일정작성을 보장한다. 매개 프로그람은 그것이 실행되는동안 프로그람의 스레드와 같은 개수의 처리기들을 배정받는다. 프로그람이 완료되면 처리기들은 다른 프로그람에 배정할수 있게 공동집결소에로 복귀한다.
- **동적일정작성법:** 실행과정에 프로쎄스의 스레드수를 변경시킬수 있다.

부하공유법

부하공유법은 가장 단순한 방법으로서 단일처리기의 환경으로부터 직접 넘겨 받은 것이다. 이 방법은 몇가지 우점을 가진다. 즉

- 부하가 처리기들에 공평하게 분산되여 진행되므로 작업을 할수 있는 동안은 처리기가 빈 상태에 놓이지 않는다.
- 중심적인 일정작성프로그람이 필요 없다. 처리기가 사용가능할 때 조작체계의 일정작성루틴은 다음 스레드를 선택하기 위하여 그 처리기에서 실행된다.
- 우선권에 기초한 방안들이나 그리고 실행경력 또는 선행하는 처리요구들을 고려하는 방안들을 포함하여 제 9 장에서 취급한 임의의 방안들을 사용하여 전역대기렬을 조직하고 접근할수 있다.

[LEUT90]에서는 부하공유법의 세가지 종류에 대하여 분석하고 있다.

- **선래선봉사(FCFS)법:** 일감이 도착하면 매개 스레드는 공유대기렬의 끝에 련속적으로 줄을 선다. 그러다가 어떤 처리기가 빈 상태로 되면 다음의 준비된 스레드를 선택하여 그것이 완료되거나 페색될 때까지 집행한다.
- **최소스레드수우선법:** 공유준비대기렬은 우선권대기렬로 조직한다. 이때 일정작성안된 스레드수가 제일 적은 일감들로부터 스레드들에 최고우선권을 준다. 우선권이 같은 일감들은 그것이 도착하는 순서에 따라 렬을 짓는다. FCFS 법에서와 같이 일정작성된 스레드를 완료하거나 페색될 때까지 실행한다.
- **선취형최소스레드수우선법:** 일정작성안된 스레드수가 제일 적은 일감들에 최고우선권을 준다. 실행중에 있는 일감보다 적은 수의 스레드를 가진 일감이 도착하면 그것은 일정작성한 일감에 속하는 스레드들을 선취한다.

저자들은 모의모형을 사용하여 넓은 범위의 일감특성지표들에 대한 분석을 진행하고 FCFS 법이 우에서 지적한 다른 두가지 방법들보다 우월하다고 보고하고 있다. 또한 저자들은 다음의 소절에서 취급하게 되는 일종의 무리일정작성방법이 부하공유법에 비하여 우월하다는것을 보고하고 있다.

부하공유법은 몇가지 결함을 가지고 있다. 즉

- 중심대기렬이 호상배제를 일으키는 식으로 호출해야 하는 기억구역을 차지하는것이다. 이때 많은 처리기들이 동시에 작업을 기다린다면 병목현상이 일어 날수 있다. 처리기수가 적을 때에는 이것이 그리 큰 문제로 되지 않는다. 그러나 다중처리기가 수십개 또는 수백개의 처리기들로 되여 있는 경우에는 병목의 위험성이 현실적인것으로 된다.
- 선취된 스레드들은 동일한 처리기에서 다시 실행할 가망이 없는것이다. 만일 매

개 처리기가 국부캐쉬를 가지고 있다면 캐쉬는 효률을 더 적게 한다.
- 모든 스레드들을 하나의 공동스레드집결소처럼 취급할 때 프로그람의 모든 스레드들이 동시에 처리기들에 접근할수 없게 하는것이다. 만일 프로그람의 스레드들사이에 높은 등급의 일치성이 요구된다면 복잡한 프로쎄스절환들에 의하여 성능을 심히 떨어 뜨릴수 있다.

이러한 잠재적인 결함들이 있음에도 불구하고 부하공유법은 현재 다중처리기들에서 가장 일반적으로 사용되는 방안들중의 하나이다.

부하공유기술을 갱신한것을 Mach 조작체계에서 사용하고 있다[BLAC90, WEND89]. 조작체계는 매개 처리기용 국부실행대기렬과 공유형 전역실행대기렬을 가지고 있다. 국부실행대기렬은 특정한 처리기에 일시 구속된 스레드들이 사용한다. 처리기는 먼저 구속된 스레드들에 구속되지 않은 스레드들보다 절대적인 우선권을 주기 위하여 국부실행대기렬을 조사한다. 구속스레드들을 사용하는 실례로서 조작체계의 요소인 프로쎄스들을 실행하는데 한개 또는 그이상의 처리기들을 독점시키는 경우를 들수 있다. 또다른 실례는 특정한 응용프로그람의 스레드들을 많은 처리기들에 분산시키는 경우이다. 이때 적당한 쏘프트웨어를 추가하면 다음에 설명하는 무리일정작성방법을 지원하게 된다.

무리일정작성법

프로쎄스모임을 처리기모임에 의하여 동시에 일정작성하는 개념은 스레드들의 사용을 전재로 한다. [JONE80]에서는 무리일정작성과 같은 개념을 고찰하고 다음과 같은 우점이 있다는것을 밝혔다.

- 밀접히 련관된 프로쎄스들을 병렬로 실행한다면 동기화의 페색을 감소시킬수 있고 필요한 프로쎄스절환을 줄일수 있으며 따라서 성능이 높아 지게 된다.
- 한번의 결정이 단번에 많은 처리기들과 프로쎄스들에 영향을 주므로 일정작성의 간접소비시간을 줄일수 있다.

Cm * 다중처리기에서는 술어 협동일정작성을 사용하고 있다[GEHR87]. 협동일정작성은 과제시행이라고 부르는 관련된 과제모임을 일정작성하는 개념에 기초하고 있다. 과제시행의 개별요소들은 매우 작고 따라서 스레드의 개념에 가깝다.

술어 무리일정작성은 한개의 프로쎄스를 구성하는 스레드들에 대한 동시적인 일정작성에 적용되였다[FEIT90b]. 응용프로그람의 어떤 부분이 실행준비를 하는 동안 다른 부분이 실행되지 못할 때 성능이 심히 떨어 지는 중간립도의 병렬응용프로그람들을 미세립도로 쪼개는것이 필요하다. 무리일정작성에 대한 필요성은 널리 인정되여 각이한 다중처리기조작체계들에 실현되였다.

무리일정작성에서 단일한 응용프로그람의 성능을 개선하는 한가지 명백한 방도는 프로쎄스절환을 최소화하는것이다. 프로쎄스의 한개의 스레드가 실행중에 있고 동일한 프로쎄스의 다른 스레드와 동기를 맞추어야 하는 점에 도달했다고 하자. 만일 그 다른 스레드가 실행중에 있지 않고 아직 준비대기렬에 있다고 하면 첫번째 스레드를 요구한 스레드를 끌어 들이기 위하여 프로쎄스절환을 어떤 다른 처리기에 의하여 할수 있을 때까지 기다리게 된다. 스레들사이에 일치성이 강한 응용프로그람에서 진행되는 그러한 절환들에 의하여 성능이 심히 떨어 지게 된다.

협동동작스레드들의 동시일정작성은 또한 자원배정에서 시간을 줄일수 있게 한다. 실례로 다중무리일정작성한 스레드들은 탐색하는 동안 읽기/쓰기연산을 페쇄함으로써 발

생하는 보충적인 간접소비시간이 없이 파일을 호출할수 있다.

무리일정작성법의 사용시에는 처리기배정에 대한 요구조건이 제기된다. 그것을 해결하는 한가지 가능성은 다음과 같다. N 개의 처리기로 M 개의 응용프로그람을 실행하되 매개 응용프로그람은 N개 또는 그보다 적은 스레드들을 가지고 있다고 하자. 그러면 매개 응용프로그람은 시간세분법을 사용하여 N개의 처리기에 사용가능시간의 1/M 을 배당할수 있다. [FEIT90a]에서는 이방법이 비효과적일수 있다는것을 지적하고 있다.

실례로 두개의 응용프로그람이 있는데 그중 한개는 네개의 스레드를 가지고 다른 한개는 한개의 스레드를 가지고 있다고 하자. 시간을 균등하게 배정하는 경우에는 처리자원의 37.5%를 랑비하게 된다. 그것은 단일스레드응용프로그람이 실행될 때 세개의 처리기들은 빈채로 남아 있기때문이다(그림 10-2 를 보라.). 만일 단일스레드응용프로그람이 몇개 있다면 이것들이 함께 어울려 처리기의 사용률을 높일수 있다. 만일 그렇게 할수 없다면 균일한 일정작성대신에 스레드수에 의해 무게를 붙인 일정작성을 한다. 그러면 네개의 스레드를 가진 응용프로그람에는 4/5 의 시간이 배당되고 한개의 스레드를 가진 응용프로그람에는 1/5 의 시간이 배당됨으로써 처리기랑비는 15%까지 떨어 진다.

그림 10-2. 네개 및 한개의 스레드를 가진 일정작성그룹의 실례[FEIT90a]

전용처리기의 할당

[TUCK89]에서 제기한 무리일정작성의 극단한 형태는 응용프로그람이 실행되는 동안 처리기그룹을 그것에 독점시키는것이다. 즉 응용프로그람을 일정작성할 때 매개 스레드를 처리기에 할당시키고 응용프로그람의 실행이 완료될 때까지 그 스레드에 처리기를 그대로 독점시키는것이다.

이 방법은 확실히 처리시간을 심히 랑비할것 같이 보인다. 만일 응용프로그람의 어떤 스레드가 다른 스레드들과의 동기를 맞추기 위하여 입출력을 기다리면서 페색된다면 그 스레드의 처리기는 빈 상태로 된다. 이때 처리기에서 다중프로그람작성은 할수 없다. 이러한 전략과 관련하여 두가지 문제에 주목할수 있다. 즉

1. 처리기가 수십 또는 수백개이고 매개 처리기가 체계가격의 적은 부분을 맡고 있는 고병렬체계에서는 처리기사용률이 효과성이나 성능의 척도로서 더는 중요하지 않다.
2. 프로그람의 수명주기동안 프로쎄스절환의 완전한 회피는 프로그람의 근본적인 속도제고의 원인으로 된다.

[TUCK89]와 [ZAHO90]에서는 두번째 문제를 지원하는 분석결과를 보고하고 있다. 그림 10-3 에는 [TUCK89]의 한가지 실험결과를 제시하였다. 저자들은 16 개의 처리기를

가진 체계에서 행렬곱하기와 고속푸리에변환(FFT)계산을 진행하는 두개의 응용프로그람을 실행시켰다. 매개 응용프로그람은 그 문제를 많은 과제들로 분할하는데 그것은 그 응용프로그람을 집행하는 스레드들에로 넘겨 진다. 프로그람들은 스레드수를 변화시키면서 사용할수 있도록 하는 방향에서 작성되였다. 많은 과제들이 정의되고 응용프로그람마다 렬을 짓게 하였다. 과제들은 대기렬로부터 선택되여 프로그람에 의하여 사용가능한 스레드들에로 넘겨 진다. 만일 과제들보다 스레드들이 더 적다면 나머지 과제들은 그냥 대기렬에 남게 되며 그것들이 할당된 과제들은 완료할 때 스레드들에 의해 선택된다. 명백히 모든 응용프로그람들은 이러한 식으로 구조화할수 없지만 많은 수값문제들과 일부 다른 응용프로그람들을 이러한 방식으로 취급할수 있다.

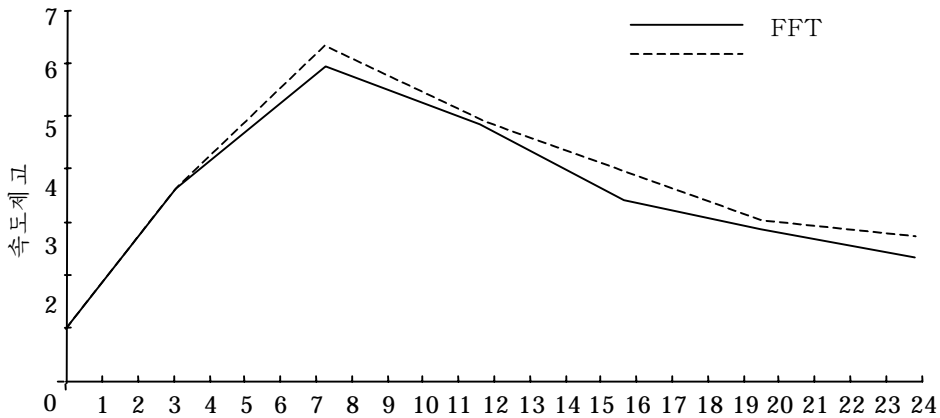



그림 10-3. 스레드수에 따르는 응용프로그람의 속도

그림 10-3에서는 매개 응용프로그람의 과제들을 진행하는 스레드수가 1에서 24까지 변화할 때 응용프로그람의 속도제고곡선을 보여 주고 있다. 실례로 두개의 응용프로그람이 각각 24 개의 스레드를 가지고 동시에 시동할 때 벌어 지는 속도제고률은 행렬곱하기에 대해서는 2.8이고 FFT에 대해서는 2.4라는것을 알수 있다. 그림에서는 매개 응용프로그람의 스레드수가 8 을 초과할 때 따라서 체계의 프로쎄스의 총수가 처리기의 개수를 초과할 때 두 응용프로그람의 성능이 현저히 떨어 진다는것을 보여 주고 있다 나아가서 스레드수가 커질수록 성능은 떨어 지는데 그것은 스레드의 선취와 재일정작성이 더 빈번히 진행되기때문이다. 이러한 과도한 선취는 중단된 스레드가 림계구간으로부터 벗어 나기를 기다리는데 걸리는 시간, 프로쎄스절환에 랑비되는시간, 비효률적인 캐쉬동작에 소비되는 시간 등을 포함하여 많은 요인들에 의하여 효률을 떨어뜨리는 결과를 초래한다.

저자들은 효과적인 전략은 능동스레드수를 체계의 처리기수로 제한하는것이라는 결론을 지었다. 만일 대부분의 응용프로그람들이 한개의 스레드를 가지거나 과제대기렬주소를 사용할수 있다고 하면 이것은 처리기자원 등의 효과적이고 효률적인 사용을 보장하게 된다.

전용처리기의 할당과 무리일정작성에서는 처리기배정문제를 지정하는것으로부터 일정작성에 착수한다. 다중처리기에서의 처리기배정문제는 단일처리기에서의 일정작성문제보다 단일처리기에서의 기억기배정문제와 더 공통성이 있다는것을 알수 있다. 문제는 임의의 주어진 시간에 얼마나 많은 처리기들을 프로그람에 할당하는가 하는것인데 이것은 임의의 순간에 얼마나 많은 페지프레임들을 주어진 프로쎄스에 할당하는가 하는것과 류

사하다. [GENH87]에서는 가상기억기의 작업모임과 비슷한 술어 활동성작업모임을 제안하였는데 그것은 응용프로그람을 순조롭게 진척시키기 위하여 처리기들에 동시에 일정작성해야 하는 최소활동(스레드)수를 말한다. 기억기관리방안에서와 같이 활동성작업모임의 모든 요소들의 일정작성을 잘못하면 처리기의 과도교체현상이 일어 날수 있다. 그것은 봉사가 요구되는 스레드의 일정작성이 봉사를 곧 받게 될 다른 스레드들의 일정작성의 해제를 야기시킬 때 발생한다. 이와 류사하게 처리기의 단편화는 일부 처리기들은 배정되고 나머지처리기들은 기다리고 있는 응용프로그람들의 요구를 충족시킬수 있도록 수적으로 충분하지 못하거나 불합리하게 조직되였을 때 어떤 처리기들이 방임상태에 있는 정황을 가리키는것이다. 무리일정작성과 전용처리기의 배정은 이러한 문제들을 회피하기 위한것이다.

동적일정작성

일부 응용프로그람들에서는 프로쎄스의 스레드수가 동적으로 변할수 있게 하는 언어와 체계도구들을 보장해 줄수 있다. 이것은 조작체계로 하여금 사용률을 개선할수 있도록 적재를 조절할수 있게 한다.

[ZAHO90]에서는 조작체계와 응용프로그람이 일정작성결정들을 하는데 사용하는 방법을 제기하고 있다. 조작체계는 처리기들을 일감들에 분배하는 책임을 지고 있다. 매개 일감은 현재 분배된 처리기들을 사용하여 이 과제들을 스레드들에 사영하는 방법으로 그것의 실행가능한 과제들의 부분모임을 집행한다. 어떤 부분모임을 실행할것인가 또한 프로쎄스가 선취될 때 어떤 스레드를 중지시킬것인가에 따라 합리적결정은 개별적인 응용프로그람들에 관계된다(아마도 실행시의 서고루틴들의 모임을 통하여). 이 방법은 모든 응용프로그람들에 적합하지 않을수 있다. 그러나 일부 응용프로그람들은 한개의 스레드를 기정사실로 할수 있고 다른 응용프로그람들은 조작체계의 이 특정한 기능을 사용할수 있도록 프로그람을 작성할수 있다.

이 방법에서 조작체계의 일정작성책임은 주로 처리기의 배정으로 제한되며 다음의 방책에 따라 진행된다. 일감이 한개 또는 처리기들을 요청할 때(일감이 첫 시간에 도착할 때나 그것의 도구가 변할 때)

1. 만일 빈 처리기들이 있다면 요청을 충족시키는데 그것들을 사용한다.
2. 그렇지 않고 요청을 내는 일감이 새로운 도착자라면 현재 한개이상의 처리기를 배정받은 임의의 일감으로부터 한개를 옮겨 놓고 그것에 한개의 처리기를 배정한다.
3. 만일 요청의 일부를 만족시킬수 없다면 그것은 프로쎄스가 그것을 위해 사용가능하게 되거나 일감이 요청을 취소할 때까지(즉 여분의 처리기들에 대한 요구가 더는 없다면) 미해결로 남아 있을것이다.

한개 또는 그이상의 처리기들이 해방(일감의 리탈을 포함하여)되자마자

4. 처리기들에 대한 현재의 만족되지 않은 요청들의 대기렬을 훑어 본다. 그리고 현재 처리기들을 가지고 있지 않는 목록에 따라(즉 기다리고 있는 새로운 도착자들모두에게) 매개 일감에 한개의 프로쎄스를 할당한다. 다음에 목록을 다시 훑어 보고 FCFS법에 따라 나머지 처리기들을 배정한다.

[ZAHO90]과 [MAJU88]의 해석에서는 응용프로그람들에서 동적일정작성법을 사용할수 있다는것을 제기하였는데 이 방법은 무리일정작성법이나 독립식처리기할당법보다 우월하다. 그러나 이 방법의 간접소비시간은 외견상 성능제고를 부정할수 있다. 동적일정작성법의 가치를 증명하자면 실제적인 체계에서 체험해 보아야 한다.

제 2 절. 실시간일정작성

배경

실시간계산분야는 점차 중요한 학문으로 되고 있다. 조작체계 특히 일정작성기는 실시간체계의 가장 중요한 요소이다. 실시간체계의 응용실례로서 연구실험조종, 기업소공정조종, 항공운수조종, 원격통신 및 국방지휘조종체계 등을 들수 있다. 다음 세대의 체계들로서는 자동착륙선, 탄성체결합형로보트조종기, 지능형제작공정체계, 우주정류소 및 해저탐사체계를 들수 있다.

실시간계산은 체계의 정확도가 계산의 론리적결과뿐아니라 결과가 산출되는 시간에 관계되는 그러한 형태의 계산으로서 정의할수 있다. 실시간프로쎄스 또는 과제 [3]가 무엇인가를 정의함으로써 실시간체계를 정의할수 있다. 일반적으로 실시간체계에서는 일부 과제가 실시간시간과제인데 이것들은 어느 정도의 긴급성을 가진다. 그러한 과제들은 외부세계에서 발생되는 사건들을 조종하거나 그것들에 반작용한다. 이 사건들은 《실제시간》으로 발생하므로 실시간과제는 그것이 관여하는 사건들을 유지할수 있어야 한다. 이때 보통 기한부를 특정과제와 련관시키는것이 가능한데 여기서 기한부는 시작시간이나 완료시간을 규정한다. 그러한 과제는 하드형과 쏘프트형으로 분류할수 있다. **하드실시간과제**는 그것의 기한부를 정하여야 하는 과제를 말한다. 그렇지 않으면 그것은 체계에 바라지 않는 손실을 주거나 치명적인 오유를 발생시킨다. **쏘프트실시간과제**는 필요는 하지만 지령에 의한것은 아닌 기한부를 가진 과제를 말한다. 즉 그것의 기한부가 지났다해도 과제를 일정작성하고 완료해도 일 없는 과제이다.

실시간과제의 다른 특성지표는 그것이 주기적인가 아니면 비주기적인가 하는것이다. **비주기적과제**는 그것을 끝내거나 시작하여야 하는 기한부를 가지거나 시작과 끝시간에 대한 제약조건을 가질수 있다. **주기적과제**인 경우에는 요구사항을《주기 T당 한번》또는《정확히 T단위 떨어 져서》라고 서술할수 있다.

실시간조작체계의 특성지표

실시간조작체계는 다섯가지 일반 령역에서 제기되는 요구사항들에 의하여 특징지을수 있다[MORG92]. 즉

- 결정론성
- 응답성
- 사용자조종
- 믿음성
- 고장완화조작

조작체계는 그것이 고정된 그리고 미리 결정된 시간들에 또는 미리 결정된 시간간격들에서 수행하는 조작들에 대하여 **결정론적**이다. 여러개의 프로쎄스들이 자원들과 처리시간을 놓고 경쟁하고 있다면 그러한 체계는 완전히 결정론적이 아니다. 실시간조작체계

[3] 참고문헌들에서 각이한 단어들을 각이한 의미로 사용하기때문에 용어에서는 문제가 있다. 특정한 프로쎄스를 반복성을 가진 실시간조것에서 동작시키는것이 일반적이다. 프로쎄스는 긴 시간동안 계속되며 그동낭네 실시간사건들에 응답하여 어떤 반복기능을 수행한다. 이 절에서는 개별적기능들을 과제라고 하자. 그러면 프로쎄스는 과제의 순서렬을 따라 진행되는 처리로서 볼수 있다. 임의의 주어 진 시간에 프로쎄스는 한개의 과제안에 예약되는데 그것은 일정작성을 해야 하는 프로쎄스/과제이다.

에서 프로쎄스의 봉사요청들은 외부사건들과 박자동기화에 기초하여 명령을 받는다. 조작체계가 결정론적으로 요청들을 만족시키는 정도는 우선 그것이 새치기들에 응답할수 있는 속도와 둘째로 요구되는 시간안에 모든 새치기들을 처리할수 있는 충분한 능력을 가지고 있는가에 관계된다.

조작체계의 결정론적인 기능을 수행할수 있는 능력을 평가하는 한가지 유효한 척도는 최대우선권을 가진 장치로부터 새치기원천이 도착하여서부터 봉사가 시작될 때까지의 최대지연시간이다. 비실시간조작체계에서는 이 지연시간이 수십～수백㎳정도일수 있다. 그러나 실시간조작체계에서는 그 지연시간이 수㎲～1㎳이내에 있어야 한다.

련관이 있으면서도 뚜렷이 구별이 되는 특성지표로서 **응답성**이 있다. 결정성은 조작체계가 새치기를 확인하기전에 얼마동안 지연되는가에 관계된다. 그러나 응답성은 새치기를 확인한후에 조작체계가 얼마동안 새치기를 봉사하는가에 관계된다. 응답성은 다음과 같은 항목들을 포함한다. 즉

1. 처음에 새치기를 조종하고 새치기봉사루틴(ISR)의 집행에 착수하는데 요구되는 시간량. 만일 새치기봉사루틴의 집행이 프로쎄스절환을 필요로 한다면 새치기봉사루틴이 현재의 프로쎄스의 문맥하에서 집행할수 있는 경우보다 지연이 더 길어 지게 된다.
2. 새치기봉사루틴을 수행하는데 요구되는 시간량. 이것은 일반적으로 하드웨어가동환경에 관계된다.
3. 새치기겹싸기의 효과. 만일 새치기봉사루틴이 다른 새치기의 도착에 의하여 또 새치기될수 있다면 봉사는 더 늦어 지게 된다.

결정론성과 응답성은 다 같이 외부사건들에 대한 응답시간을 발생시킨다. 응답시간 요구사항은 실시간체계들에서 결정적인것이다. 그것은 그러한 체계들이 인원, 장치, 체계밖의 자료흐름에 부과된 시간동기의 요구를 만족시켜야 하기때문이다.

사용자조종은 일반적으로 보통의 조작체계들에 비하여 실시간조작체계에서 훨씬 더 범위가 넓다. 대표적으로 비실시간조작체계에서 사용자는 조작체계의 일정작성기능에 대한 조종을 할수 없고 다만 사용자들을 한개이상의 우선권부류로 그룹을 묶는것과 같은 넓은 범위의 안내를 보장한다. 그러나 실시간체계에서는 사용자가 과제우선권에 대한 미세립도의 조종을 할수 있게 하는것이 필수적이다. 실시간체계는 또한 사용자가 페지화나 프로쎄스교체방법의 사용과 같은 특성지표들을 명시할수 있게 하며 어떤 프로쎄스들을 항상 주기억기에 상주시켜야 하는가, 어떤 디스크이송알고리듬들을 사용하겠는가, 각이한 우선권대역에서 프로쎄스들에 어떤 권한을 부여하겠는가 등을 명시할수 있게 한다.

믿음성은 비실시간체계들보다 실시간체계에서 대표적으로 훨씬 더 중요하다. 비실시간체계에서 순간적인 고장은 고장난 처리기가 수리되거나 교체될 때까지 봉사준위를 낮추는결과를 초래한다. 그러나 실시간체계는 실시간적으로 사건들에 대응하거나 그것들을 조종한다. 성능의 손실이나 저하는 재정적손실로부터 중요한 설비의 손상, 지어는 인간의 사망에 이르기까지의 치명적결과를 가져 올수 있다.

다른 측면에서 보면 실시간조작체계와 비실시간조작체계사이의 차이는 등급상의 차이이다. 실시간체계는 각이한 고장형식들에 대응할수 있도록 설계되여야 한다. **고장완화조작**은 가능한 많은 기능들과 자료를 보존하는 견지에서 본 고장에 대한 체계의 능력이라고 말할수 있는 특성지표이다. 실례로 전통적인 UNIX 체계는 핵심부의 자료가 잘못되였을 때 고장통보문을 체계조종탁에 내고 후에 고장분석을 하기 위하여 기억기의 내용을

디스크에 쏟아 넣고 체계의 집행을 끝낸다. 그것과는 대조적으로 실시간체계는 제기된 문제를 바로 잡거나 실행을 계속하면서 그것의 효과를 최소화하려고 한다.대표적으로 체계는 사용자나 사용자프로쩨스에게 수정작업을 하면서 축소된 봉사준위에서 조작을 계속한다음 통보한다. 전원을 꺼야 할 사건인 경우에는 파일과 자료의 일관성을 유지하도록 해야 한다.

고장완화조작의 중요한 측면은 안전성이다. 모든 과제의 기한부들을 만족시키는것이 불가능한 경우에 체계가 일부 중요한 과제의 기한부를 항상 만족시키지 못한다하더라도 가장 중요하고 최고우선권을 가진 과제들의 기한부를 만족시키게 된다면 실시간체계는 안정하다.

우에서 지적한 요구사항들을 만족시키자면 현재의 실시간조작체계들은 대표적으로 다음과 같은 체모를 갖추어야 한다[STAN89]. 즉

- 고속프로쩨스 또는 스레드절환
- 작은 규모(련관된 최소기능성을 가진)
- 외부새치기들에 즉시 응답할수 있는 능력
- 신호기들, 신호들, 사건들과 같은 프로쩨스간 통신도구들을 가진 다중과제처리
- 빠른 속도로 자료를 축적할수 있는 전용 순서파일들의 사용
- 새치기가 금지되는 시간간격의 최소화
- 고정된 시간동안 과제를 지연시키거나 과제들을 잠시정지/다시시작하기 위한 수단
- 특수한 경보 및 시간만기수단

실시간체계의 심장부는 단기과제일정작성기이다. 그러한 일정작성기에서 공평하고 최소화된 평균응답시간은 가장 중요한것이 아니다. 중요한것은 모든 하드실시간과제들을 그것들의 기한부에 의하여 완료(시작)하며 가능한 많은 쏘프트웨어실시간과제들을 그것들의 기한부에 따라 완료(시작)하는것이다.

대부분의 현재 실시간조작체계들은 기한부들을 직접 취급할수 없게 되여 있다. 그대신 그것들은 기한부에 가까와 올때 과제가 빨리 일정작성될수 있도록 실시간과제들에 가능한 응답하게끔 설계되여 있다. 이러한 관점으로부터 실시간응용들에서는 여러가지 조건하에서 수㎳～수분의 1 ㎳의 범위에서 결정론적인 응답을 요구한다. 군용항공기와 같은 중요한 응용들에서는 보통 10~100 ㎲범위의 제한조건을 준다[ATLA89].

과제들에 가능한 응답하게끔 설계되여 있다. 이러한 관점으로부터 실시간응용들에서는 여러가지 조건하에서 수㎳～수분의 1 ㎳의 범위에서 결정론적인 응답을 요구한다. 군용항공기와 같은 중요한 응용들에서는 보통 10~100 ㎲범위의 제한조건을 준다[ATLA89].

그림 10-4 에서는 가능성의 스펙트르를 보여 주고 있다. 단순한 순환형일정작성기를 사용하는 선취식일정작성기는 그림 10-4 ㄱ에서 보는바와 같이 다음의 세분시간을 기다리기 위해 대기렬에 추가된다. 이 경우에 일정작성시간은 일반적으로 실시간응용들에 접수될수 없다. 그대신 비선취식일정작성기에서는 실시간과제들에 보다 높은 우선권을 주는 우선권일정작성기구를 사용할수 있다. 이 경우에 준비된 실시간과제는 현재의 프로쩨스가 폐색되거나 실행완료되자마자 곧 일정작성된다(그림 10-4 ㄴ). 이때 만일 완만하고 우선권이 낮은 과제가 림계시간에서 실행되고 있었다면 수 s 의 지연을 발생시킬수 있다.

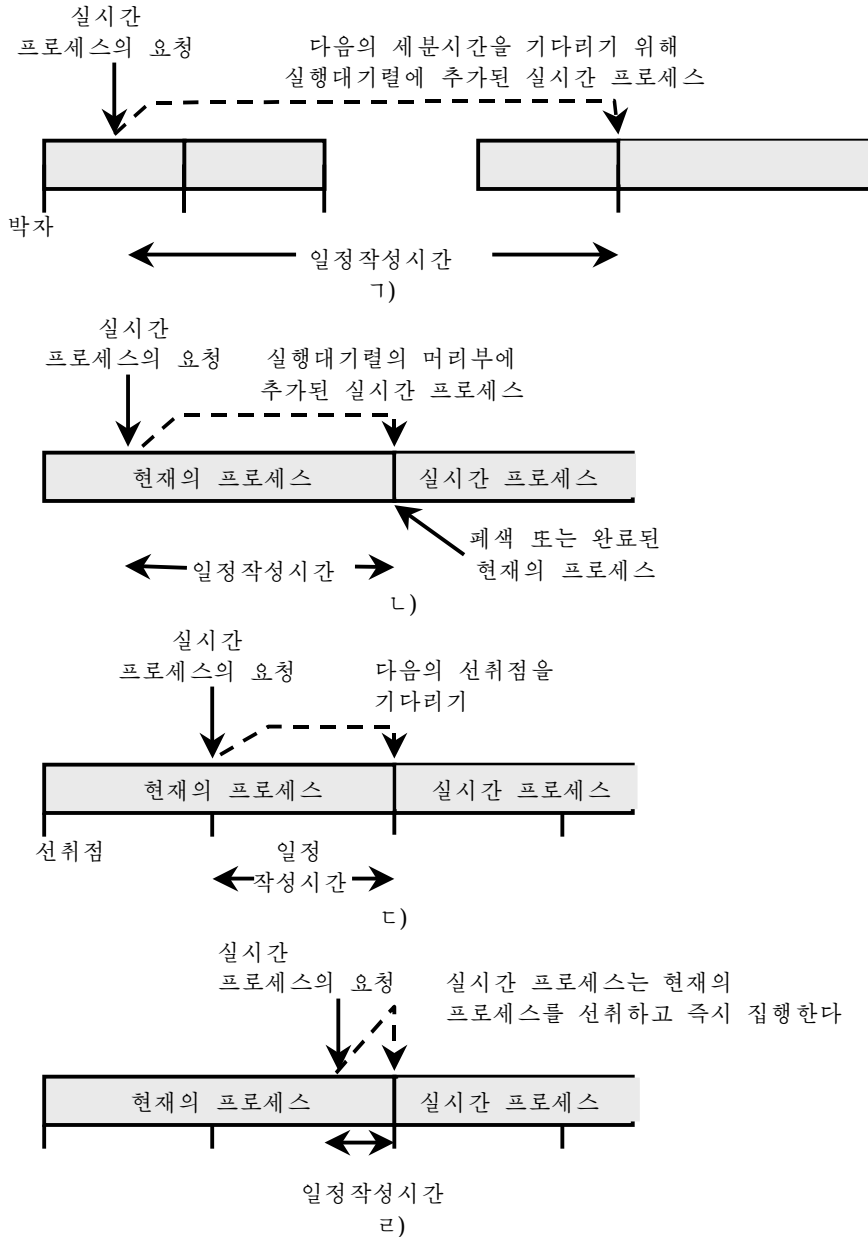



그림 10-4. 실시간프로세스의 일정작성
ㄱ-순환선취식일정작성기, ㄴ-우선권구동비선취식일정작성기
ㄷ-우선권구동선취점선취식일정작성기, ㄹ-즉시선취식일정작성기

이 방법역시 접수될수 없다. 보다 가능성있는 방법은 우선권들을 시계에 기초한 새치기들과 조합한 방법이다. 선취점들이 규칙적인 시간간격으로 발생된다. 선취점이 발생할 때 보다 높은 우선권을 가진 과제가 선취된다. 이것은 조작체계핵심부의 한 부분인 과제들의 선취를 포함한다. 그러나 지연은 수㎳의 정도에 달할수 있다(그림 10-4 ㄷ). 이 마지막 방법이 일부 실시간응용에는 적합하지만 신청된 많은 응용들에서는 적합하지

못하다. 그러한 경우들에는 때때로 즉시선취법이라고 부르는 방법을 적용한다. 이때 조작체계는 그것이 림계코드페색구역에 있지 않다면 새치기에 거의 즉시에 응답한다. 그러면 실시간과제의 일정작성지연시간을 100 ㎲ 또는 그보다 더 적은 값으로 줄일수 있다.

실시간일정작성

실시간일정작성분야는 콤퓨터과학에서 가장 활발히 벌어 지고 있는 연구분야의 하나이다. 이 절에서는 실시간일정작성의 여러가지 방법들을 개괄하고 두가지 부류의 보편적인 일정작성알고리듬들을 고찰한다.

실시간일정작성알고리듬들을 조사하여 보면 [RAMM94]에서는 각이한 일정작성방법들이 (1) 체계가 일정작성가능성에 대한 분석을 하는가 안하는가 (2) 만일 해석을 한다면 그것을 정적으로 하는가 아니면 동적으로 하는가 (3) 분석결과에 따라 그자체가 일정을 작성하는가 아니면 실행시에 어떤 과제들이 배분되는가에 따라서 계획하는가에 관계된다는것을 밝히고 있다. 이러한 고찰에 기초하여 알고리듬들을 두가지 부류로 갈라볼수 있다. 즉

- **정적표구동방법:** 배분의 가능한 일정에 대한 정적인 분석을 진행한다. 분석결과에 대하여 실행시간에 과제를 언제 집행하기 시작하여야 하는가를 결정하는 일정을 작성한다.
- **정적우선권구동식선취형방법:** 역시 정적분석을 하지만 일정작성은 하지 않는다. 분석은 오히려 전통적인 우선권구동식선취형일정작성기에서 사용할수 있도록 과제들에 우선권들을 할당하기 위하여 진행한다.
- **동적계획법에 의한 방법:** 가능성은 실행에 착수하기전에 비직결식(정적으로)으로가 아니라 실행시에 결정(동적으로)된다. 도착하는 과제는 그것이 시간적제한조건들을 만족할 가능성이 있는 경우에만 실행을 위해 접수된다. 가능성분석에 대한 하나의 결과에 의하여 일정을 작성하며 이 과제를 언제 배분하겠는가를 결심하는데 사용할수 있도록 계획을 한다.
- **동적최상노력방법:** 가능성분석은 진행하지 않는다. 체계는 모든 기한부들을 만족시키기 위하여 노력하며 기한부를 지키지 못한 임의의 시동된 과제를 포기하도록 한다.

정적표구동식일정작성법은 주기적인 과제들에 적용할수 있다. 분석에 필요한 입력은 주기적인 도착시간, 실행시간, 주기적인 마감기한부 및 매개 과제의 상대적인 우선권으로 이루어 진다. 일정작성기는 주기적인 모든 과제들의 요구사항들을 만족시킬수 있는 일정을 작성하려고 한다. 이것은 예측할수 있는 방법이지만 유연성이 없는 방법이다. 그것은 임의의 과제의 요구사항이 달라지면 일정작성을 다시 하여야 하기 때문이다. 최단기한부우선법이나 다른 주기적기한부수법들(다음에 설명하는)은 이러한 부류에 속하는 대표적인 일정작성알고리듬들이다.

정적우선권구동식선취형일정작성법에서는 대부분의 비실시간다중프로그람처리체계들에서 공통적인 우선권구동식선취형일정작성기구들을 사용한다. 비실시간체계에서는 각이한 인자들을 우선권을 결정하는데 사용할수 있다. 실례로 시분할체계에서 프로쎄스의 우선권은 그것이 처리기위주인가 아니면 입출력위주인가에 따라서 변한다. 실시간체계에서 우선권할당은 매개 과제와 관련된 시간적제약조건들에 관계된다. 이 방법의 한가지 실례로서 속도단조알고리듬(다음에 설명하는)을 들수 있는데 이것은 과제들의 주기의 길이에 기초하여 그것들에 정적인 우선권들을 할당한다.

동적계획법에 의한 일정작성법에서는 과제가 도착한후에가 아니라 그것의 실행을 시작하기전에 이미 일정작성된 과제들과 도착한 새로운 과제들을 포함하는 일정을 창조한다. 만일 도착한 새로운 과제의 기한부들이 만족되고 현재 일정작성된 과제가 기한부를 지키지 못한데로부터 새로운 과제를 일정작성할수 있다고 하면 새로운 과제를 수행하기 위하여 일정작성이 수정된다.

동적최상노력일정작성법은 현재 상업적으로 사용가능한 많은 실시간체계들에 사용하고 있는 방법이다. 어떤 과제가 도착하면 체계는 과제의 특성지표에 기초하여 우선권을 할당한다. 최초기한부일정작성법과 같은 기한부일정작성법의 일부 형태가 대표적으로 사용되고 있다. 대표적으로 과제들은 비주기적이고 따라서 정적인 일정작성분석은 불가능하다. 이러한 형태의 일정작성에서 기한부에 이르거나 과제가 완료될 때까지 박자동기제약조건이 만족되는지 안되는지를 알수 없다. 이것이 이러한 형태의 일정작성법들의 주요한 결함이다. 이 방법들의 우점은 실현하기가 쉬운것이다.

기한부일정작성

현대 대부분의 실시간조작체계들은 실시간과제들을 가능한 빨리 시작하고 이로부터 빠른 새치기처리와 과제배분을 뚜렷이 살릴 목적에서 설계되였다. 사실상 이것은 실시간체계들을 평가하는데서 특별히 중요한 척도로 되지는 못한다. 실시간응용들은 일반적으로 순수한 속도에 관심을 가지는것이 아니라 동적인 자원도구들과 충돌들, 과부하들의 처리 그리고 하드웨어나 쏘프트웨어적인 오유들의 조건에서도 지내 일찍도 아니고 지내 늦게도 아닌 가장 긴요한 시간에 완료되는(시작하는) 과제에 오히려 관심을 가진다.

최근년간에 실시간과제의 일정작성을 위한 보다 위력하고 합리적인 방법들이 많이 제기되였다. 이 모든 방법들은 매개 과제에 대한 추가적인 정보에 기초하고 있다. 매개 정보에 대한 가장 일반적인 형태로서 다음의 정보를 사용할수 있다. 즉

- **준비시간:** 과제를 실행하기 위해 준비하는 시간이다. 반복적이거나 주기적인 과제인 경우에 이것은 실제적으로 미리 알려져 있는 시간순서이다. 비주기적인 과제인 경우에 이 시간은 미리 알수 있다. 다시 말하면 조작체계는 과제가 실제로 언제 준비되였는가를 알아 차릴수 있다.
- **시작기한부:** 과제를 시작하여야 하는 시간이다.
- **완료기한부:** 과제를 완료해야 하는 시간이다. 대표적인 실시간응용에서는 시작기한부들이나 완료기한부들을 가질뿐 두 기한부들을 모두 가지지는 않는다.
- **처리시간:** 과제를 끝까지 실행하는데 필요한 시간이다. 어떤 경우에는 이것이 제시되지만 다른 경우에는 조작체계가 그것의 지수평균을 측정한다. 아직 다른 일정작성체계들에서는 이 정보를 사용하지 않는다.
- **자원요구사항:** 과제가 실행되는 동안 과제에 요구되는 자원(처리기가 아닌)들의 모임이다.
- **우선권:** 과제의 상대적인 중요도를 재는 단위이다. 기한부를 지키지 못하면 고장이라고 보는 체계에서 하드실시간과제들은 (절대적인) 우선권을 가진다. 만일 체계가 어떤 일에 관계 없이 실행을 계속하여야 하는 경우에 하드 및 쏘프트실시간과제들은 일정작성기의 안내자로서 상대적인 우선권들을 할당할수 있다.
- **부분과제구조:** 과제는 의무적인 부분과제와 선택적인 부분과제로 분해할수 있다. 의무적인 부분과제만이 하드기한부를 가진다.

다음에 어떤 과제를 일정작성하겠는가 그리고 어떤 종류의 선취권을 허용하겠는가를

고려할 때 실시간일정작성기능에는 몇가지 차원이 있다. 선취권전략이 제시되고 시작기한부나 완료기한부를 사용할 때 최초의 기한부를 가진 과제를 일정작성하는 방책은 그것들의 기한부들을 지키지 못한 과제들의 비률을 최소화한다는것을 보여 주고 있다 [BUTT99, HONG89, PANW88]. 이러한 결론은 단일처리기와 다중처리기의 환경에서 모두 성립한다.

표 10-2. 두개의 주기적인 과제에 대한 실행분포

프로쎄스	도착시간	집행시간	마감기한부
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
⋮	⋮	⋮	⋮
B(1)	0	25	50
B(2)	50	25	100
⋮	⋮	⋮	⋮

다른 중요한 설계문제는 선취권에 대한 문제이다. 시작기한부들이 명시될 때에는 비선취식일정작성기가 적합하다.이러한 경우에 실시간과제의 책임은 그것의 외부적인 부분과 림계적인 부분의 실행을 완료한후에 그자체를 페색함으로써 다른 실시간시작기한부들을 만족시키도록 하는것이다. 이것이 그림 10-4 ㄴ의 경우이다. 완료기한부들을 가진 체계에서는 선취식전략(그림 10-4 ㄷ나 ㄹ)이 가장 적합하다. 실례로 과제 X 가 실행중이고 과제 Y 가 준비되여 있다고 하면 X 와 Y 가 다 그것들의 완료기한부들을 만족시키도록 하는 유일한 방도는 X 를 선취하고 Y 를 끝까지 집행하고 다음에 X 를 끝까지 다시 집행하는것이다.

완료기한부들을 가진 주기적인 과제들을 일정작성하는 실례로서 두개의 수감부 A 와 B 로부터 들어 오는 자료들을 수집하여 처리하는 체계를 고찰하자. 수감부 A 로부터 들어 오는 자료를 수집하기 위한 기한부는 매번 20ms 를 만족시켜야 하고 수감부 B 에 대해서는 매번 50ms 를 만족시켜야 한다. A 의 매개 표본자료를 처리하는데 조작체계의 간접소비시간을 포함하여 10ms 가 걸리고 B 의 매개 표본자료를 처리하는데는 25ms 가 걸린다. 표 10-2 에 두 과제들의 실행분포를 요약하여 제시하였다.

콤퓨터는 매개 10ms 마다 일정작성결정을 할 능력을 가지고 있다. 이러한 환경에서 우선권일정작성방안을 사용하려고 한다고 하자. 이때의 결과를 그림 10-5 의 첫 두개의 시간선도에서 보여 주고 있다. 만일 A 가 보다 높은 우선권을 가진다고 하면 B 의 첫 실례에서는 두개의 10ms 의 구간에 해당한 20ms 의 처리시간밖에 주어 지지 않으므로 끝까지 실행되기전에 기한부에 도착하여 실패한다. 만일 B 가 우선권이 더 높다면 A 는 첫번째 기한부에서 실패한것이다. 마지막 시간선도에서는 최소기한부일정작성법을 사용한 경우를 보여 주고 있다. 시간 t=0 에서 A1 과 B1 이 도착한다. A1 이 맨 처음의 기한부를 가지고 있으므로 그것이 먼저 일정작성된다. A1 이 완료되면 B1 에 처리기가 배정된다.

t=20 에서 A2 가 도착한다. A2 가 B1 보다 기한부가 더 먼저이므로 A2 가 끝까지 실행될 수 있도록 B1 은 새치기된다. 그러다가 B1 은 t=30 에서야 다시 계속된다. t=40 에서 A3 이 도착한다. 그러나 B1 의 마감기한부가 더 먼저이므로 B1 이 계속 실행되며 t=45 에서 완료된다. 그다음에 A3 이 처리기를 배정받아 t=55 에서 끝낸다.

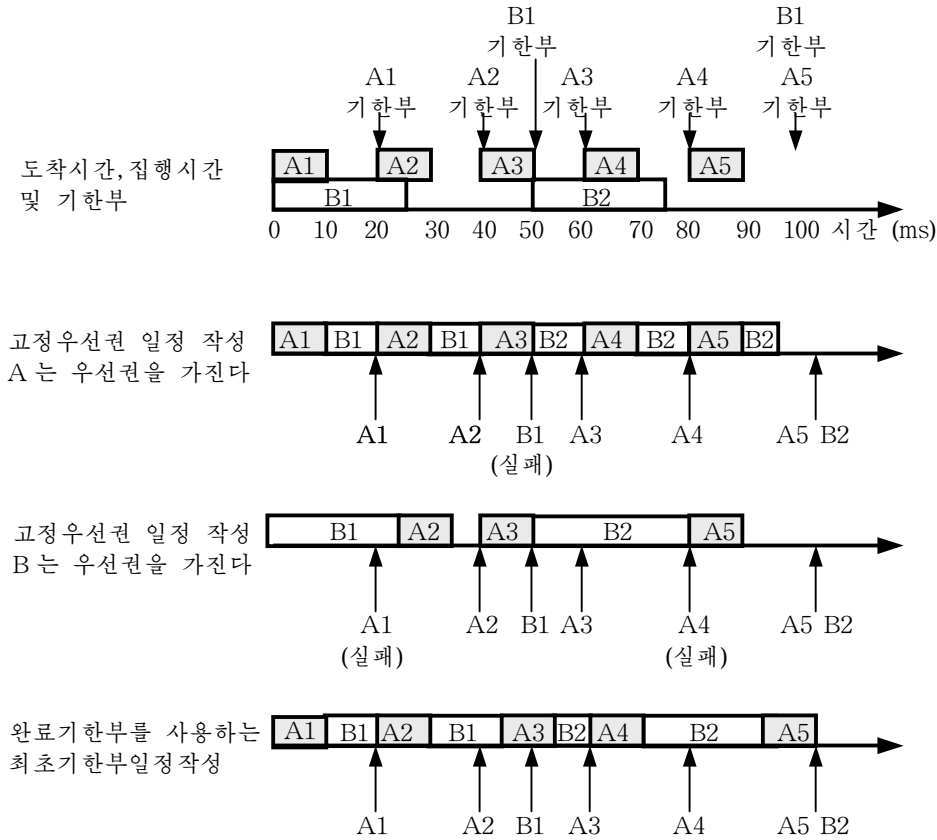



그림 10-5. 완료기한부를 가진 주기적인 실시간과제의 일정작성

이 실례에서는 임의의 선취점에서 가장 가까운 기한부를 가진 과제에 우선권을 주는 일정작성법에 의하여 체계의 모든 요구사항들을 만족시킬수 있다. 과제가 주기적이고 예측가능한것이므로 정적표구동식일정작성방법을 사용한다.

이제 시작기한부들을 가진 비주기적인 과제들을 취급하는 방안을 보기로 하자. 그림 10-6 의 웃부분에서는 실행시간이 각각 20ms 인 다섯개의 과제로 되여 있는 실례에 대한 도착시간들과 시작기한부들을 보여 주고 있다. 표 10-3 에서는 다섯개의 과제들의 실행분포를 요약하여 보여 주고 있다.

간단한 방안은 최초의 기한부를 가진 준비된 과제를 일정작성하며 그 과제를 끝까지 실행하는것이다. 이 방법을 그림 10-6 의 실례에서 사용한 경우에 과제 B 가 즉시적인 봉사를 요구하였음에도 불구하고 봉사를 거절당한데 대하여 주목하자. 이것이 비주기적인 과제 특히 시작기한부를 가진 과제를 취급할 때 제기되는 현상이다. 이 방책을 개량하여 과제가 준비되는 시간에 앞서 알수 있게 한다면 성능을 개선한다. 바로 자연적인 빈 시간이 있는 최초기한부방법이라고 부르는 이 방책은 다음과 같이 동작한다.

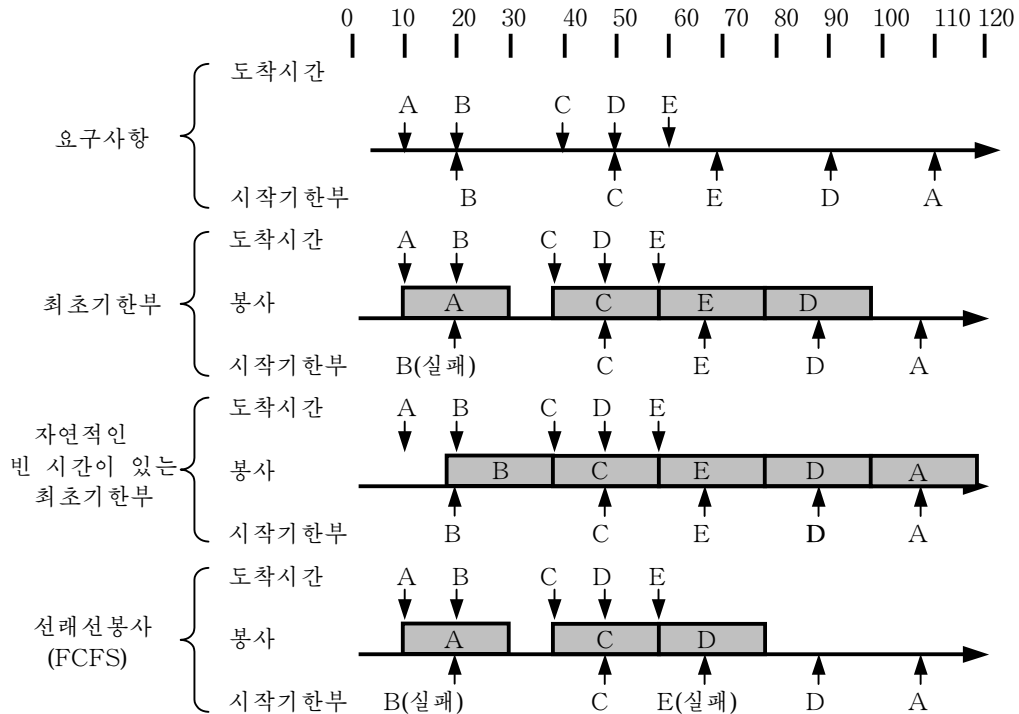



그림 10-6. 시작기한부들을 가진 비주기적인 실시간과제의 일정작성

표 10-3. 다섯개의 비주기적인 과제들에 대한 실행분포

프로쎄스	도착시간	집행시간	시작기한부
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

항상 최초의 기한부를 가진 적중한 과제를 일정작성하여 그 과제를 끝까지 실행하도록 한다. 적중한 과제는 준비되지 않을수 있고 이것은 준비된 과제들이 있다고 하여도 처리기를 빈상태로 남아 있게 한다. 우의 실례에서 과제 A가 준비된 유일한 과제임에도 불구하고 체계는 그것의 일정작성을 그만 둔다는것에 주의를 돌린다. 결과는 처리기가 최대의 효률로 사용되지 않음에도 불구하고 일정작성의 요구사항들을 만족한다는것이다. 마지막으로 비교를 위하여 FCFS방책을 제시하였다. 이 경우에 과제 B와 E는 그것들의 기한부들을 만족시키지 못한다.

속도단조일정작성법

주기적인 과제들에 상반되는 다중과제일정작성문제를 해결하기 위한 보다 유망한 한가지 방법이 바로 속도단조일정작성(RMS)법이다. 이 방안은 [LIU73]에서 처음으로 제기되였는데 최근에야 보편화되였다[BRI99, SHA94]. RMS에서는 과제들의 기간에 기초하여 그것들에 우선권들을 할당한다.

그림 10-7 에서는 주기적인 과제들에 대한 파라메터들을 설명하고 있다. 과제의 주기T는 과제의 한 실례가 도착하여서부터 과제의 다음 실례가 도착할 때까지의 시간량이다. 과제의 속도(Hz)는 그것의 주기(s)의 역수로 간단히 구할수 있다. 실례로 주기가 50 ㎳인 과제는 20Hz 의 속도로 발행한다. 대표적으로 과제주기의 끝은 또한 비록 일부 과제들이 시간적으로 보다 짧은 기한부들을 가질수 있다고 하더라도 과제의 하드기한부이다. 집행(또는 계산)시간C는 과제가 매번 발행하는데 걸리는 처리시간량이다. 단일처리기체계에서 집행시간은 주기보다 크지 말아야 한다(C ≦ T)는것이 명백하다. 만일 주기적인

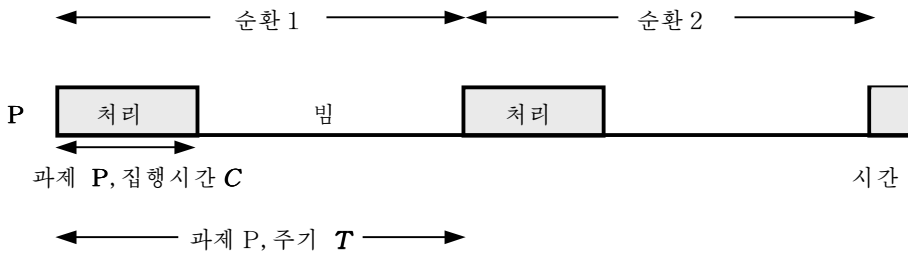



그림 10-7. 주기적인 과제의 시간선도

과제가 항상 끝까지 실행된다면 다시 말하여 과제의 실례가 불충분한 자원들때문에 봉사를 거절당하는 일이 없다면 이 과제에 대한 처리기의 사용률은 $U = C/T$ 이다. 실례로 과제의 주기가 80 ㎳이고 집행시간이 55 ㎳라고 하면 그것의 처리기사용률은 55/80=0.6875 이다.

RMS 에서 최고우선권과제는 최단주기를 가진것이고 두번째 최고우선권과제는 두번째 최단주기를 가진것이며 이런식으로 계속되여 나간다. 한개이상의 과제가 집행에 사용할수 있을 때 최단주기를 가진것이 먼저 봉사받는다. 과제들의 우선권은 그것들의 속도의 함수로서 곡선을 그리면 결과는 단조증가하는 함수로 된다(그림 10-8). 여기로부터 속도단조일정작성법이라는 이름이 붙게 되였다.

주기적일정작성알고리듬의 효과성을 측정하는 한가지 기준은 그것이 모든 하드기한부들을 만족시키는것을 담보하는가 안하는가 하는것이다. 매개가 고정된 주기와 실행시간을 가지는 N개의 과제를 고찰하자. 이때 모든 기한부들을 만족시킬수 있게 하자면 다음의 부등식이 성립하여야 한다. 즉

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots \frac{C_n}{T_n} \le 1 \qquad (10\text{-}1)$$

개별적과제들의 처리기사용률의 합은 값 1 을 넘을수 없는데 그것은 처리기의 총 사용률에 해당한다. 식 10-1 은 완전한 일정작성알고리듬을 성과적으로 할수 있는 과제수의 한계를 준다. 임의의 특정한 알고리듬에서 그 한계는 보다 작을수 있다. RMS법에서는 다음의 부등식이 성립한다는것을 알수 있다. 즉

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \le n(2^{1/n} - 1) \qquad (10\text{-}2)$$

표 10-4 에서는 이 웃한계값들을 일부 보여 주고 있다. 과제수가 증가할 때 일정작성의 한계는 $\ln 2 \approx 0.693$ 에로 다가 간다.

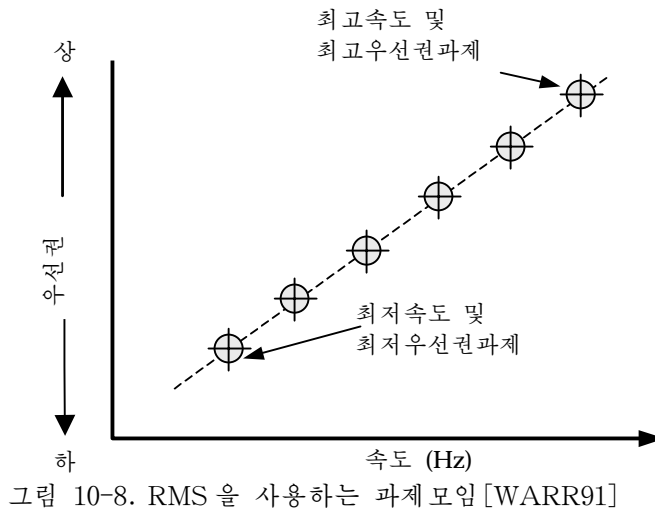



그림 10-8. RMS 을 사용하는 과제모임[WARR91]

실례로 $U_I = C_I/T_i$ 인 세개의 주기적인 과제인 경우를 고찰하자. 즉

- 과제 P1: $C_1 = 20$; $T_1 = 100$; $U_1 = 0.2$
- 과제 P2: $C_2 = 40$; $T_2 = 150$; $U_2 = 0.267$
- 과제 P3: $C_3 = 100$; $T_3 = 350$; $U_3 = 0.286$

이 세개의 과제들에 대한 총 사용률은 0.2 + 0.267 + 0.286 = 0.753 이다. RMS 법을 사용할 때 이 세가지 과제들의 일정작성능력의 웃한계는

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} \leq 3(2^{1/3} - 1) = 0.779$$

이다. 세개의 과제들에 요구되는 총 사용률은 RMS 법인 경우에 웃한계보다 작아야 하므로(0.753 < 0.779) RMS 법을 사용한다면 모든 과제들은 성공적으로 일정작성된다.

또한 식 10-1 의 웃한계는 최초기한부일정작성법에서 성립한다는것을 알수 있다. 이때 보다 큰 총체적인 처리기사용률을 얻을수 있고 따라서 보다 많은 주기적인 과제들에 최초기한부일정작성법을 적용할수 있다. RMS 법도 역시 공업적인 응용분야들에 널리 적용되였다. 이에 대하여 [SHA91]에서는 다음과 같이 설명하고 있다. 즉

1. 실천에서는 성능차이가 적다. 식 10-2 의 웃한계는 지내 파도한것이고 실천적으로 사용률이 보통 90 %정도로 높아 진다.
2. 대부분의 하드실시간체계들은 또한 그리 중요하지 않은 현시기들과 그리고 하드실시간과제들에 RMS 법을 사용하지 않는 처리기시간을 관찰하기 위하여 보다 낮은 우선권준위에서 집행할수 있는 자체시험프로그람과 같은 쏘프트실시간요소들을 가지고 있다.
3. RMS 법에 의하여 쉽게 안전성을 보장할수 있다. 체계가 과부하와 순간적인 오유들로 인하여 모든 기한부들을 만족시킬수 없을 때 본질적인 과제들의 기한부들은 과제들의 이 부분모임을 일정작성가능하다는것을 담보할것을 요구한다. 정적우선권할당방법에서는 본질적인 과제들에 상대적으로 높은 우선권들을 확실히 부여하는것이 필요하다. 이것은 RMS 법에서 본질적인 과제들이 짧은 주기들을

가지도록 구성하거나 본질적과제들이 계산에 대하여 RMS 우선권들을 변경하는 방법으로 수행된다. 최초기한부일정작성법에서 주기적인 과제들의 우선권은 한 주기로부터 다른 주기로 넘어갈 때 변경된다. 이것은 본질적인 과제들이 그것들의 우선권들을 만족시키도록 담보하는것을 더욱 어렵게 한다.

표 10-4. RMS 의 웃한계값

n	$n(2^{1/n}-1)$
1	1.0
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
⋮	⋮
∞	ln2 = 0.693

제 3 절 Linux 의 일정작성

LINUX 는 쏘프트실시간처리를 위한 두가지 일정작성부류를 혼합하여 제 9 장 제 3 절에서 설명한 전통적인 UNIX 에 구축한것이다. LINUX 에서 적용하고 있는 세가지 부류의 일정작성법은 다음과 같다. 즉

- **SCHED-FIFO**: 선입선출실시간스레드법
- **SCHED_RR**: 순환실시간스레드법
- **SCHED_OTHER**: 기타 비실시간스레드법

매개 부류에서 다중우선권을 사용할수 있는데 SCHED_OTHER 부류보다 실시간부류들의 우선권들의 준위가 더 높다. FIFO 스레드법들에서 다음의 규칙들을 적용한다. 즉

1. 체계는 다음의 경우들을 제외하고는 집행중에 있는 FIFO 스레드를 새치기할수 없다.
   ㄱ) 보다 우선권이 높은 다른 FIFO 스레드는 준비상태로 된다.
   ㄴ) 집행중에 있는 FIFO 스레드는 입출력과 같은 사건을 기다리면서 페색된다.
   ㄷ) 집행중에 있는 FIFO 스레드는 자의대로 처리기를 포기하고 기본지령 sched_yield 를 호출한다.
2. 집행중에 있는 FIFO 스레드가 새치기될 때 그것은 우선권과 관련된 대기렬에 배치된다.
3. FIFO 스레드가 준비될 때 그리고 만일 그 스레드가 현재 집행중에 있는 스레드보다 우선권이 더 높을 때에는 현재 집행중에 있는 스레드는 선취되고 보다 높은 우선권을 가진 FIFO 스레드가 집행된다. 만일 한개이상의 스레드가 보다 높은 우선권을 가진다면 가장 오래동안 기다리고 있던 스레드가 선택된다.

SCHED_RR 방책은 SCHED_FIFO 방책과 류사한데 다른 점은 매개 스레드와 관련되는 시간분배몫을 더해 주는것이다. SCHED_RR 스레드가 자기의 시간할당몫들만 집행하였을 때 그것은 중단되며 우선권이 동일하거나 더 높은 실시간스레드가 실행을 위해 선택된다.

[COMP98]에서 인용한 그림 10-9 는 FIFO 와 RR 일정작성법의 차이를 설명하는 실례이다. 어떤 프로그람이 그림 10-9 ㄱ에서 보여 주는것과 같이 할당된 세가지 상대우선권들을 가진 네개의 스레드를 가진다고 하자. 또한 기다리는 모든 스레드들에 현재의 스레드가 기다리거나 완료될 때 실행하기 위하여 준비되거나 스레드가 실행중에 있는 동안은 보다 높은 우선권을 가진 스레드가 깨여 나지 못한다고 하자. 그림 10-9 ㄴ에서는 모든 스레드들이 SCHED-FIFO 부류에 있다는것을 보여 주고 있다. 스레드 D 는 그것이 기다리거나 완료될 때까지 실행된다. 다음 스레드 B 와 C 가 동일한 우선권을 가진다고 하더라도 스레드 B 는 그것이 스레드 C 보다 더 오래동안 기다렸기때문에 먼저 시동된다. 스레드 B 는 그것이 기다리거나 완료될 때까지 집행된다. 다음에 스레드 C 가 그것이 기다리거나 완료될 때까지 집행된다. 결국 스레드 A 는 맨 마지막에 집행된다.

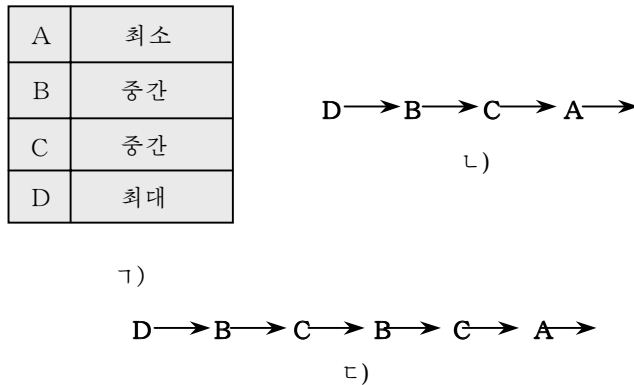



그림 10–9. LINUX 의 일정작성의 실례
ㄱ-상대스레드 우선권, ㄴ-FIFO 일정작성법의 흐름
ㄷ-RR 일정작성법의 흐름

그림 10-9 ㄷ에서는 모든 스레드들이 SCHED_RR 부류에 있을 때의 흐름실례를 보여 주고 있다. 스레드 D 는 그것이 기다리거나 완료될 때까지 집행된다. 다음에 스레드 B 와 C 가 시간세분된다. 그것은 B 와 C 가 동일한 우선권을 가지고 있기때문이다.

마지막에 스레드 A 가 집행된다. 마지막 일정작성부류는 SCHED_OTHER 이다. 이 부류의 스레드는 실시간스레드들이 집행을 위해 준비되지 않고 있을 때에만 집행된다. SCHED_OTHER 부류에서는 제 9 장 제 3 절에서 설명한 전통적인 UNIX 일정작성알고리듬이 사용된다.

제 4 절. UNIX SVR4 의 일정작성

UNIX SVR4 에서 사용하는 일정작성알고리듬은 이전의 UNIX 체계(제 9 장 3 절에서 설명한)에서 사용한 일정작성알고리듬을 수정완성한것이다. 새로운 알고리듬은 실시간프로쎄스들에 가장 높은 우선권을, 핵심부방식의 프로쎄스들에 다음으로 높은 우선권을 그리고 시간공유프로쎄스들이라고 부르는 다른 사용자방식의 프로쎄스들에 가장 낮은 우선권을 주도록 설계하였다.

SVR4 에서 실현한 두가지 중요한 변경내용은 다음과 같다. 즉

1. 선취가능한 정적우선권일정작성기와 세가지 우선권부류로 나눈 160 개의 우선권 준위모임의 도입
2. 선취점들의 삽입. 기본핵심부는 선취되지 않으므로 새치기없이 끝까지 실행시켜

야 하는것은 처리단계들로 분할한다. 처리단계들사이에서 선취점이라고 부르는 안전한 대목들은 핵심부가 어디에서 처리를 안전하게 새치기하며 새로운 프로쎄스를 일정작성할수 있는가를 확정한다. 안전한 대목들은 모든 자료구조들이 갱신되여 일관성이 보장되거나 신호기를 통하여 폐쇄되는 코드구역으로 정의된다.

그림 10-10에서는 SVR4에서 정의한 160개의 우선권준위들을 보여 주고 있다. 매개 프로쎄스들은 세가지 우선권부류의 하나에 속하도록 정의하고 그 부류안에서는 우선권준위를 할당한다. 우선권부류들은 다음과 같다. 즉

- **실시간(159～100):** 이 우선권준위들에 속하는 프로쎄스들은 임의의 핵심부나 시분할프로쎄스에 앞서 실행하도록 선택하는 책임을 진다.
- **핵심부(99～60):** 이 우선권준위들에 속하는 프로쎄스들은 임의의 시분할프로쎄스에 앞서 실행하도록 하면서도 실시간프로쎄스들까지 기다리도록 선택하는 책임을 진다.
- **시분할(59-0):** 실시간응용들이 아닌 사용자응용들을 위한 우선권이 가장 낮은 프로쎄스들이 여기에 속한다.

그림 10-11 에서는 SVR4 에서 일정작성을 어떻게 실현하는가를 보여 주고 있다. 배분대기렬은 매개 우선권준위와 관련되며 주어진 우선권준위의 프로쎄스들은 순환식으로 집행된다. 비트사용벡토르는 매개 우선권준위용으로 한개의 비트를 포함하고 있는데 그 비트는 비여 있지 않는 대기렬을 가진 임의의 우선권준위에 대하여 1 로 설정된다. 실행중에 있는 프로쎄스가 폐색, 시간세분만기 또는 선취로 인하여 실행상태를 벗어날 때마다 일정작성기는 dgactmap 를 검사하여 최고우선권이 가장 높은 비여 있지 않는 대기렬로부터 준비프로쎄스를 배분한다. 또한 정의된 선취점에 도달할 때마다 핵심부는 kprunrun 이라는 기발을 검사한다. 만일 설정되여 있으면 그것은 적어도 한개의 실시간프로쎄스가 준비상태에 있고 그것이 최고우선권실시간준비프로쎄스보다 더 낮은 우선권을 가지고 있다면 핵심부가 현재의 프로쎄스를 선취한다는것을 가리킨다.

우선권 부류	전역값	일정작성순서
실시간	159 ⋮ 100	먼저 ↓
핵심부	99 ⋮ 60	↓
시분할	59 ⋮ 0	↓ 마감

그림 10–10. SVR4 의 우선권부류

시분할부류안에서 프로쎄스의 우선권은 가변적이다. 일정작성기는 프로쎄스가 시간량자를 다 사용할 때면 매번 프로쎄스의 우선권을 감소시키며 프로쎄스가 사건이나 자원으로 인하여 폐색될 때에는 그것의 우선권을 증가시킨다. 시분할프로쎄스에 배정된 시간량자는 그것의 우선권에 관계되는데 우선권 0 의 100ms 로부터 우선권 59 의 10ms 범위에 있다. 매개 실시간프로쎄스는 고정된 우선권과 고정된 시간량자를 가진다.

제 5 절. Windows 2000 의 일정작성

Windows 2000(W2K)은 고속의 대화환경에서 또는 봉사기의 역할을 하면서 단일 사용자의 요구에도 가능한 응답하도록 설계되여 있다. W2K 에서는 우선권준위체계가 유연한 일정작성기를 실현하고 있는데 매개 우선권준위내에서는 순환일정작성법을 적용하고 있으며 일부 준위에서는 현재의 스레드활동성에 기초하여 동적가변우선권을 가진다.

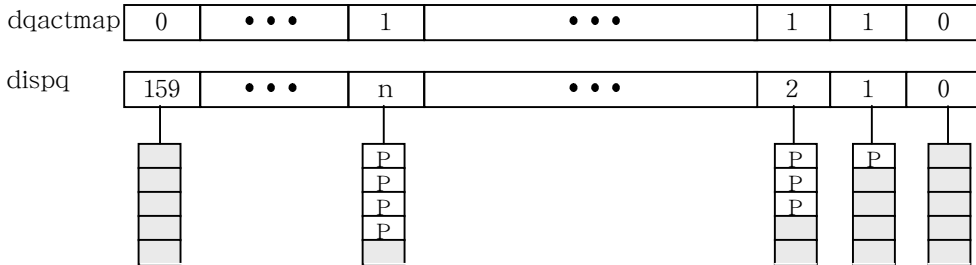



그림 10-11. SVR4 의 배분대기렬들

프로쎄스 및 스레드우선권

W2K 의 우선권들은 두가지 대역과 부류 즉 실시간부류와 가변부류로 조직된다. 즉시적인 반응을 요구하는 스레드들은 실시간부류에 속하는데 여기에는 통신과 실시간과제들과 같은 기능들이 포함된다.

총체적으로 W2K 에서는 우선권구동선취식일정작성기를 사용하기때문에 실시간우선권을 가진 스레드들은 다른 스레드들에 비하여 선행권을 가진다. 단일처리기에서는 현재 집행중에 있는 스레드보다 더 높은 우선권을 가진 스레드가 준비되면 보다 낮은 우선권을 가진 스레드는 선취되고 처리기에는 보다 높은 우선권을 가진 스레드가 차례진다.

우선권들은 두가지 부류에서 서로 조금 다르게 처리된다(그림 10-12). 실시간우선권부류에서 모든 스레드들은 언제나 변하지 않는 고정된 우선권을 가진다. 주어진 우선권준위에서 모든 능동스레드들은 순환대기렬에 들어 있다. 가변우선권부류에서 스레드의 우선권은 초기에 할당된 값으로부터 시작하여 스레드의 수명주기동안에 증가 또는 감소될수 있다. 이때 매개 우선권준위에 FIFO 대기렬이 있지만 프로쎄스는 가변우선권부류에서 다른 하나의 대기렬에로 이주할수 있다. 그러나 우선권준위 15 에 속하는 스레드는 준위 16 에로 또는 실시간부류의 다른 임의의 준위에로 승진시킬수 없다.

가변우선권부류에서 스레드의 초기우선권은 두가지 량 즉 프로쎄스기준우선권과 스레드기준우선권에 의하여 결정된다. 프로쎄스객체의 한가지 속성은 프로쎄스기준우선권인데 이것은 0～15 중에서 어느 한개의 값을 취할수 있다. 프로쎄스객체와 관련된 매개 스레드객체는 프로쎄스의것과 상대적인 스레드기준우선권을 표시하는 스레드기준우선권 속성을 가진다. 스레드의 기준우선권은 그것의 프로쎄스의것과 같거나 프로쎄스의것보다 2 준위 더 높거나 더 낮을수 있다. 실례로 프로쎄스가 기준우선권 4 를 가진다면 그것의 스레드의 우선권은 기준우선권으로서 –1 이고 따라서 그 스레드의 초기우선권은 3 이다.

일단 가변우선권부류의 스레드가 활성화되였다면 스레드의 동적우선권이라고 부르는 실제우선권은 주어진 경계들사이에서 변동될수 있다. 동적우선권은 스레드의 기준우선권의 낮은 한계 아래로 떨어 질수 없으며 또한 15 를 결코 초과할수 없다. 그림 10-13 에서는 그러한 실례를 보여 주고 있다. 객체는 기준우선권속성 4 를 가진다. 이 프로쎄스객체와 관련된 매개 스레드객체는 2~6 사이의 초기우선권을 가져야 한다. 매개 스레드에 대한 동적우선권은 2~15 의 범위에서 변동될수 있다. 만일 스레드가 그것의 현재 시간

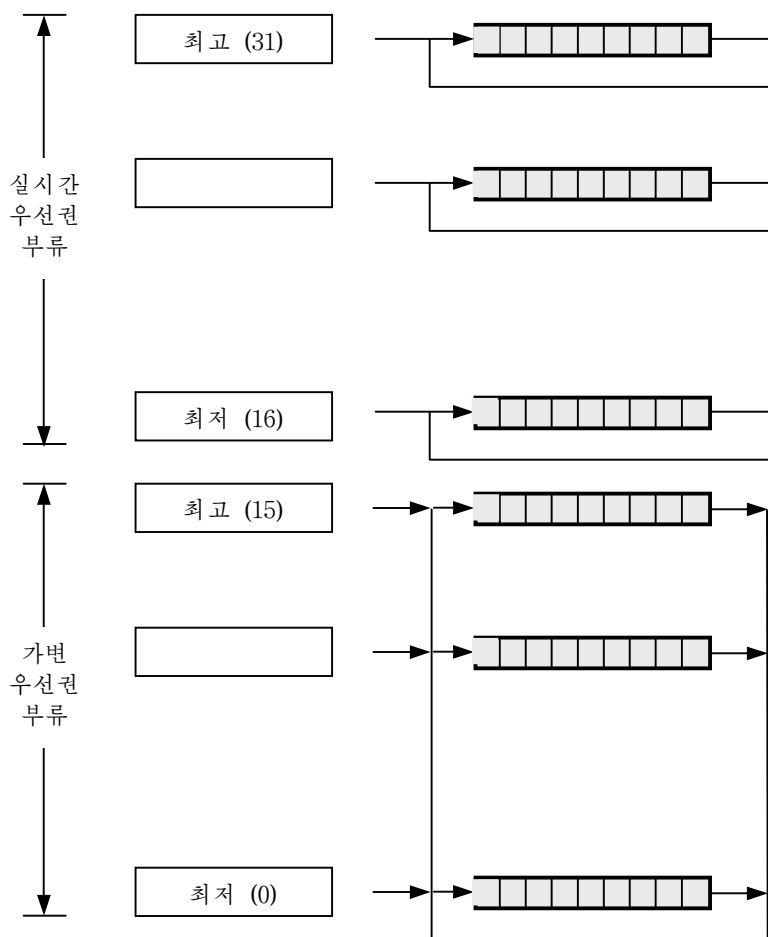



그림 10-12 . Windows NT 의 스레드배분우선권

량자를 다 써 버린 결과에 새치기되였다면 W2K 집행부는 그것의 우선권을 낮춘다. 만일 스레드가 입출력사건을 기다리도록 새치기된다면 W2K 집행부는 그것의 우선권을 높인다. 이때 처리기위주의 스레드들은 보다 낮은 우선권들에 치우치고 입출력위주의 스레드들은 보다 높은 우선권들에 치우친다. 입출력위주의 스레드들인 경우에 집행부는 다른 형태의 입출력(실례로 디스크입출력)보다 대화형기다림(실례로 건반이나 화면기다림)을 위한 우선권을 더 크게 높인다. 따라서 대화형스레드들은 가변우선권부류내에서 가장 높은 우선권을 가진다.

다중처리기의 일정작성

W2K 가 단일처리기에서 실행될 때 최고우선권스레드는 항상 그것이 사건을 기다리지 않는이상 능동상태에 있다. 만일 최고우선권을 가지는 한개이상의 스레드가 있다면 우선권준위에 있는 모든 스레드들사이에서 순환법을 공유한다. N 개의 처리기들을 가진 다중처리기체계에서 $(N\text{-}1)$개의 최고우선권스레드들은 항상 능동상태에 있으면서 $(N\text{-}1)$개의 여유처리기에서 배타적으로 실행된다. 낮은 우선권을 가진 나머지 스레드들은 한개의 나머지 처리기를 공유한다. 실례로 만일 세개의 처리기가 있다면 두개의 최고우선권스레드들은 두개의 처리기에서 실행되며 나머지 모든 스레드들은 나머지 처리기에서

실행된다.

그림 10-13. Windows NT 의 우선권관계의 실례

우에서 지적한 규칙은 스레드의 처리기계렬의 속성에 의하여 영향을 받는다. 만일 스레드가 집행할 준비가 되였다해도 사용가능한 처리기들이 처리기계렬모임에 없다면 그 스레드는 기다리게 하고 집행부는 다음에 사용가능한 스레드를 일정작성한다.

요약, 기본용어 및 복습문제

밀결합형다중처리기에서 다중처리기는 동일한 주기억기에 접근한다. 이러한 구성에서 일정작성구조는 얼마간 더 복잡하다. 실례로 주어진 프로쎄스는 그것이 완전한 수명주기들만 동일한 처리기에 할당할수 있거나 그것이 실행상태에 들어갈 때마다 매번 임의의 처리기에로 배분될수 있다. 성능연구결과는 다중처리기체계에서 각이한 일정작성알고리듬들사이의 차이가 그리 크지 않다는것을 보여 주고 있다.

실시간프로쎄스나 과제는 어떤 프로쎄스나 기능, 콤퓨터체계와 떨어 진 외부사건모임과 관련되는것이고 그것은 외부환경과 효과적으로 그리고 정확히 상호작용하기 위하여 한개 또는 그이상의 기한부들을 만족시켜야 한다. 실시간조작체계는 실시간프로쎄스들을 관리할수 있는 능력을 가진 조작체계이다. 이와 관련하여 일정작성알고리듬에 대한 전통적인 기준들은 적용하지 않는다. 오히려 기본인자는 기한부들의 집합이다. 이러한 정황에서는 선취권과 상대적인 기한부들에 대한 반응에 크게 기대를 걸고 있는 알고리듬들이 적합하다.

기본용어

비주기적과제	립도	실시간조작체계
기한부일정작성법	하드실시간과제	실시간일정작성
결정론적인 조작체계	부하공유	응답성
고장완화조작	주기적과제	쏘프트실시간과제
무리일정작성법	속도단조일정작성법	스레드일정작성

복습문제

1. 동기화립도의 다섯가지 서로 다른 범주들을 들고 간단히 정의하시오.

2. 스레드일정작성의 네가지 수법들을 들고 간단히 정의하시오.
3. 부하공유의 세가지 변종을 들고 간단히 정의하시오.
4. 하드 및 쏘프트실시간과제들사이의 차이는 무엇인가?
5. 주기적 및 비주기적 실시간과제들사이의 차이는 무엇인가?
6. 실시간조작체계에 대한 요구사항의 다섯가지 일반적령역을 들고 간단히 정의하시오.
7. 실시간일정작성알고리듬의 네가지 부류를 들고 간단히 정의하시오.
8. 과제에 대한 정보의 어떤 항목들이 실시간일정작성에 유익한가?

참 고 문 헌

[WEND89]에서는 다중처리기의 일정작성방법들을 흥미있게 취급하고 있다. 론문집들인 [KRIS94], [STAN93], [LEE93] 및 [TILB91]에는 실시간조작체계들과 일정작성법들에 대한 중요한 기사들이 실려 있다. [ZEAD97]에서는 SVR4의 실시간일정작성기의 성능을 분석하고 있다.

KRIS94 Krishna, C., and Lee, Y., eds. "Special Issue on Real-Time Systems." *Proceed ings of the IEEE,* January 1994.

LEE93 Lee, Y., and Krishna, C., eds. *Readings in Real-Time Systems. Los* Alamitos, CA: IEEE Computer Society Press, 1993.

STAN93 Stankovic, J., and Ramamritham, K., eds. *Advances in Real-Time Systems.* Los Alamitos, CA: IEEE Computer Society Press, 1993.

TILB91 Tilborg, A., and Koob, G.. eds. *Foundations of Real-Time Computing: Scheduling and Resource Management.* Boston: Kluwer Academic Publishers, 1991.

WEND89 Wendorf, J.; Wendorf, R.; and Tokuda, H. "Scheduling Operating System Processing on Small-Scale Microprocessors." *Proceedings, 22nd Annual Hawaii International Conference on System Science,* January 1989.

ZEAD97 Zeadally, S. "An Evaluation of the Real-Time Performance of SVR4.0 and SVR4.2." *Operating Systems Review,* January 1977.

련 습 문 제

1. 실행분포가 표 10-5 와 같은 세가지 주기적과제모임을 고찰하자. 이 과제모임에 대하여 그림 10-5 와 류사한 일정작성방안들을 작성하시오.

표 10-5. 련습문제 1 의 실행분포

프로쎄스	도착시간	집행시간	마감기한부
A(1)	0	10	20
A(2)	20	10	40
⋮	⋮	⋮	⋮
B(1)	0	10	50
B(2)	50	10	100
⋮	⋮	⋮	⋮
C(1)	0	15	50
C(2)	50	15	100
⋮	⋮	⋮	⋮

2. 실행분포가 표 10-6 과 같은 다섯가지 비주기적과제모임을 고찰하자. 이 과제모임에 대한 그림 10-6 과 류사한 일정작성방안들을 작성하시오.

3. 이 문제는 속도단조일정작성을 위한 식 10-2 가 일정작성을 하는데 충분한 조건으로 되지만 필요한 조건으로는 되지 못한다. 즉 식 10-2 가 만족되지 않는다고 하더라도 때로는 성공적인 일정작성이 가능하다는것을 보여 준다.

ㄱ) 다음의 독립적이고 주기적인 과제들로 된 과제모임을 고찰하자.

표 10-6. 련습문제 2 의 실행분포

프로쎄스	도착시간	실행시간	시작기한부
A	10	20	100
B	20	20	30
C	40	20	60
D	50	20	80
E	60	20	70

- 과제 P_1: $C_1 = 20$; $T_1 = 100$
- 과제 P_2: $C_2 = 30$; $T_2 = 145$

속도단조일정작성법을 사용하여 이 과제들을 성공적으로 일정작성할수 있는가?

ㄴ) 이제 다음의 과제를 모임에 첨가하자. 즉

- 과제 P_3: $C_3 = 68$; $T_3 = 150$

이때 식 10-2 가 만족되는가?

ㄷ) 앞에서 지적한 세개의 과제들의 첫 실례가 시간 $t = 0$ 에서 도착한다고 하자. 그리고 매개 과제의 첫 기한부가 다음과 같다고 하자. 즉

$D_1 = 100$; $D_2 = 145$; $D_3 = 150$

속도단조일정작성법을 사용하면 세가지 모든 기한부들이 만족되겠는가? 앞으로 매개과제를 반복하기 위한 기한부들은 어떠한가?

4. 8 개의 처리기를 가진 다중처리기는 20 개의 테프구동기들을 가지고 있다. 집행을 완료하는데 매개가 최대로 네개의 테프구동기가 필요한 체계에 수많은 일감들이 있다. 매개 일감은 먼저 장시간동안 세개의 테프구동기만을 가지고 실행하다가 그것의 연산이 끝날 무렵에 짧은 시간동안 네번째 테프구동기를 요구한다고 가정하자. 또한 그러한 일감들이 끝없이 보충된다고 하자.

ㄱ) OS 의 일정작성기는 사용가능한 네개의 테프구동기들이 없다면 기동하지 않는다고 하자. 일감을 시작할 때 네개의 구동기들은 즉시 할당되고 일감이 끝날 때까지 해방되지 않는다. 이때 곧 실행중에 있을수 있는 일감의 최대수는 얼마인가? 이 방책의 결과로서 빈 상태로 남아 있을수 있는 테프구동기의 최대 및 최소수는 각각 얼마인가?

ㄴ) 테프구동기의 사용률을 개선하고 동시에 체계의 교착을 회피하기 위한 다른 방책을 생각해 보자. 곧 진행중에 있을수 있는 일감의 최대수는 얼마인가? 놀고 있는 테프구동기의 수의 한계는 얼마인가?

제 5 편. 입출력 및 파일

제 5 편의 중심

조작체계설계에서 가장 번잡스러운 부분은 입출력기능과 파일관리체계이다. 입출력에서 기본문제는 성능이다. 입출력기능은 콤퓨터의 성능을 결정하는 중요한 요소이다. 콤퓨터체계의 내부동작을 보면 처리기속도가 계속 높아 지고 있지만 아직은 단일처리기가 충분한 속도를 내지 못하고 있는 시점에서 SMP구성에 의하여 작업속도를 높이기 위한 다중처리기들이 제공되고 있다. 내부기억접근속도도 처리기속도와 같은 증가률은 못되지만 역시 높아 지고 있다. 그럼에도 불구하고 1준위, 2준위 또는 그이상준위의 내부캐쉬를 능숙하게 사용함으로써 처리기속도와 보조를 맞추도록 주기억접근시간을 관리하고 있다. 그러나 입출력은 특히 디스크기억기의 경우에는 큰 성능상의 도전에 부딪치고 있다.

파일체계에서도 역시 성능이 문제이다. 믿음성과 보안과 같은 다른 설계요구사항도 크게 제기된다. 사용자의 견지에서 파일체계는 조작체계의 가장 중요한 개념이다. 사용자는 파일들에 대한 빠른 접근을 요구하지만 파일들이 잘못 되지 않게 하고 그것들의 허가와 접근을 안전하게 책임 진다.

제 5 편의 안내

제11장. 입출력관리와 디스크일정작성

제11장에서는 먼저 입출력기억장치들과 조작체계의 입출력기능의 조직을 개괄한다. 다음에 성능을 개선하기 위한 각이한 완충전략들을 취급한다. 이 장의 나머지부분에서는 디스크의 입출력에 대하여 설명한다. 응답시간을 개선하기 위하여 디스크접근의 물리적특성들을 사용할수 있도록 다중디스크요청들을 일정작성할수 있는 방도들을 고찰한다. 다음에 성능과 믿음성을 개선하기 위한 디스크배렬의 사용에 대하여 검토한다. 마지막으로 디스크캐쉬에 대하여 고찰한다.

제12장. 파일관리

제12장에서는 여러가지 형태의 파일조직방법을 고찰하고 파일관리와 파일호출과 관련된 조작체계문제들을 조사한다. 또한 자료의 물리적 및 론리적조작방법들을 취급한다. 그리고 대표적인 조작체계가 사용자들에게 주는 파일관리와 관련된 봉사에 대하여 설명한다. 다음에 파일관리체계의 한 부분인 특수한 기구들과 자료구조들에 대하여 고찰한다.

제 11 장. 입출력관리와 디스크일정작성

조작체계설계에서 복잡한 대상은 입출력이다. 그것은 입출력장치들과 그 장치들의 응용프로그람들이 매우 다양하기때문이다. 따라서 일반성과 일관성을 보장하는것은 어려운 문제이다.

이 장에서는 먼저 입출력장치들과 입출력기능의 조직에 대하여 간단히 언급한다. 그 내용은 콤퓨터구성방식에 속하는것으로서 조작체계의 시점에서 보면 입출력검사에 대한 단계에 해당한다.

다음 절에서는 설계목적과 입출력기능을 구성하는 방법을 비롯하여 조작체계설계문제들을 고찰한다. 그다음에는 입출력완충에 대하여 고찰한다. 조작체계에 의하여 제공되는 기본입출력봉사의 하나가 바로 완충기능이다. 이것은 전반적인 성능을 높여 준다.

다음 절들에서는 자기디스크입출력에 대하여 전면적으로 고찰한다. 현재의 체계들에서 입출력형식은 가장 중요하게 취급되며 사용자가 직접 감촉하는 기본동작부분이다. 이 장에서는 디스크입출력동작에 대한 모형을 개발하는것으로부터 시작하여 성능을 높이는데 적용할수 있는 몇가지 수법들을 고찰한다.

이 장의 부록에서는 자기디스크와 빛기억기를 비롯하여 2차고속기억장치들에 대한 특성지표들을 요약한다.

제 1 절. 입출력장치

제1장에서도 언급한것처럼 콤퓨터체계와 동반하여 입출력에 사용되는 외부장치들은 세가지 부류로 가를수 있다. 즉

- **사람이 읽을수 있는 형 :** 콤퓨터사용자와 통신하는데 적합한 부류. 실례로서 인쇄기들, 비데오현시장치말단들, 건반 기타 마우스와 같은것들을 들수 있다.
- **기계가 읽을수 있는 형 :** 전자설비들과 통신하는데 적합한 부류. 실례로서 디스크 및 테프구동기들, 수감기들, 조종기들과 수행기구들을 들수 있다.
- **통신형 :** 원격장치들과 통신하는데 적합한 부류. 실례로서 수자식선로구동기들과 모뎀들을 들수 있다.

부류마다 큰 차이들이 있으며 매개 부류안에서도 실질적인 차이들이 있다. 그 차이들은 다음과 같다. 즉

- **자료속도 :** 자료이송속도들사이에 몇자리씩 차이들이 있을수 있다. 그림 11-1에서는 몇가지 실례들을 제시하고 있다.
- **응용프로그람 :** 설치되는 장치의 사용은 조작체계와 보장되는 편의프로그람들의 쏘프트웨어와 방책들에 영향을 준다. 실례로 파일용으로 사용되는 디스크는 파일관리쏘프트웨어의 지원을 요구한다. 가상기억조직의 페지들을 림시보관하는데 쓰이는 디스크는 가상기억하드웨어와 쏘프트웨어에 의존한다. 더 나아가서 이러한 응용프로그람들은 디스크일정작성알고리듬에 영향을 미친다(이 장의 뒤부분에서 설명한다.). 또 다른 실례로서 말단은 사용자나 체계관리자에 의해 사용될수 있다. 이때에는 조작체계의 각이한 특권준위들과 각이한 우선권들이 동반된다.

- **조종의 복잡성 :** 인쇄기는 상대적으로 단순한 조종대면부를 요구한다. 그러나 디스크는 훨씬 더 복잡하다. 이러한 차이들이 조작체계들에 미치는 영향은 다음절에서 언급하겠지만 장치를 조종하는 입출력모듈의 복잡성에 따라 어느 정도까지는 해소된다.

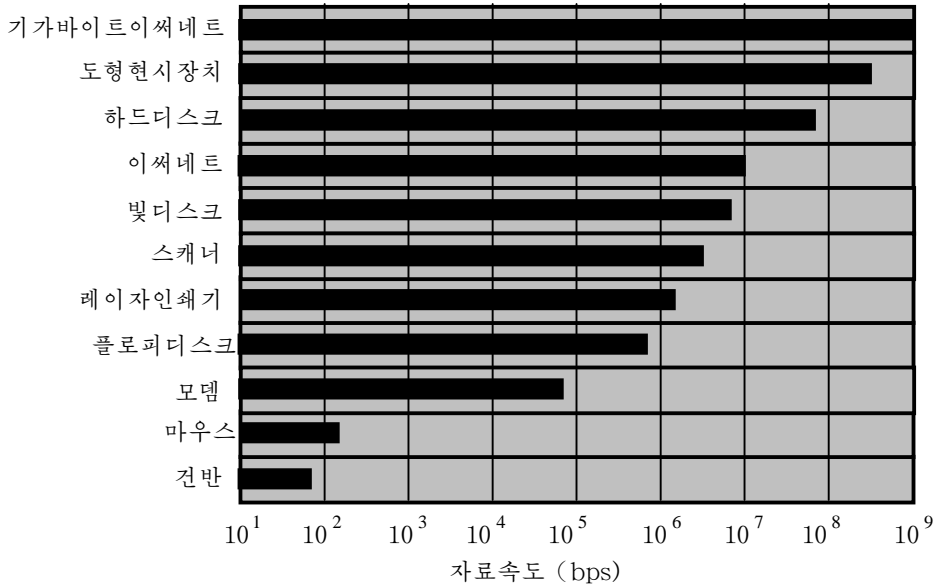



그림 11-1. 대표적인 입출력장치들의 자료속도

- **이송단위 :** 자료는 바이트 또는 문자들의 흐름(실례로 말단입출력)이나 보다 큰 블로크(실례로 디스크입출력)로 이송될수 있다.
- **자료표현 :** 각이한 자료부호화방법들은 문자코드와 기우성방법에서의 차이를 포함하여 각이한 장치들에서 사용되고 있다.
- **오유조건들 :** 오유들의 성질, 오유통보방식, 오유의 영향 및 응답들의 사용가능범위는 장치마다 다르다.

이러한 다양성은 조작체계의 관점과 사용자프로쎄스의 관점으로부터 입출력에 대한 통일적이면서도 일관성 있는 방법을 확립하는데 난관을 조성한다.

제 2 절. 입출력기능의 조직

제1장 제7절에서는 입출력을 수행하기 위한 세가지 수법들을 요약하였다. 즉

- **프로그람식입출력 :** 처리기는 프로쎄스를 대신하여 입출력지령을 입출력모듈에 발행한다. 다음에 그 프로쎄스는 수행하기전에 조작이 완료되기를 기다린다.
- **새치기구동식입출력 :** 처리기는 프로쎄스를 대신하여 입출력지령을 발행하고 다음명령들을 계속 집행하다가 마지막명령이 작업을 끝냈을 때 입출력모듈에 의하여 새치기된다. 만일 그 프로쎄스가 입출력의 완료를 기다릴 필요가 없다면 다음명령들은 동일한 프로쎄스안에 존재할수 있다. 한편 프로쎄스는 새치기에 걸려 중단되며 다른 작업이 수행된다.

- **직접기억기접근(DMA) :** DMA모듈은 주기억기와 입출력모듈사이에서 자료의 교환을 조종한다. 처리기는 자료블로크의 이송을 위한 요청을 DMA모듈에로 보내고 블로크가 완전히 이송된후에만 새치기된다.

표 11-1에서는 이 세가지 수법들사이의 관계를 보여 주고 있다. 대부분의 콤퓨터체계들에서 DMA는 조작체계에서 제공해야 할 주되는 이송형태이다.

입출력기능의 발전

콤퓨터체계들이 발전함에 따라 개별적요소들의 복잡성과 다양성도 증가하였다. 지금에 와서 입출력기능에서는 이것이 보다 더 뚜렷해 지고 있다. 그 발전단계들은 다음과 같이 요약할수 있다. 즉

1. 처리기는 직접 주변장치를 조종한다. 이것은 단순한 극소형처리기로 조종되는 장치들에서 실현되고 있다.
2. 조종기 또는 입출력모듈을 추가한다. 처리기는 새치기가 없는 프로그람식입출력을 사용한다. 이 단계를 거쳐 처리기는 외부장치대면부들의 구체적인 세부들로부터 어느 정도 분리된다.
3. 2단계와 비슷한 구성이지만 새치기를 새롭게 적용한다. 처리기는 입출력조작이 수행되기를 기다리는 시간을 랑비하지 않으며 따라서 효률이 올라 간다.
4. 입출력모듈은 DMA를 거쳐 기억기를 직접 조종한다. 그것은 이송시작과 마감을 제외하고는 처리기의 개입이 없이 기억기의 안팎으로 자료블로크를 옮길수 있다.
5. 입출력모듈은 입출력을 위한 전용의 명령들을 가진 개별적인 처리기로 되도록 기능을 강화한다. 이때 중앙처리장치(CPU)는 주기억기의 입출력프로그람을 실행하도록 입출력처리기에 지시한다. 입출력처리기는 처리기의 간섭이 없이 이 명령들을 불러 내여 집행한다. 이것은 처리기로 하여금 입출력동작들의 순서를 규정하게 하며 모든 순서가 다 수행된 때에라야 새치기를 일으키게 한다.
6. 입출력모듈은 자체의 국부기억기를 가지며 사실상 그자체가 하나의 콤퓨터로 된다. 이러한 방식에서는 소형의 처리기환경에서 입출력장치들의 큰 규모의 모임을 조종할수 있다. 이러한 방식을 공통적으로 사용하자면 대화말단들과의 통신을 조종해야 하였다. 입출력처리기는 말단들을 조종하는데 참가하는 많은 과제들에 주의를 돌린다.

표 11-1. 입출력수법

	새치기없음	**새치기사용**
처리기를 거쳐 입출력대 기억기이송	프로그람식입출력	새치기구동식입출력
직접입출력대 기억기이송		직접기억기접근(DMA)

이와 같이 입출력기능이 발전풍부화되는 과정에 더욱더 많은 입출력기능들이 처리기의 참가없이 수행되게 되였다. 중앙처리기는 입출력과 관련된 과제들에서 점차 해방되고 따라서 성능은 높아 졌다. 마지막 두 단계(5, 6단계)에서는 프로그람을 집행할수 있는 입출력모듈의 개념을 도입함으로써 근본적인 변화가 일어 났다.

전문용어에 대한 주의: 4～6단계에서 지적한 모든 모듈들에 대해서는 직접기억기접근(DMA)이라는 용어가 적합하다. 왜냐하면 이러한 모든 형태가 입출력모듈에 의해 주

기억기에 대한 직접조종을 포함하고 있기때문이다. 또한 5단계에서 입출력모듈은 자주 **입출력통로**라고 부르고 있고 6단계에서 입출력모듈은 **입출력처리기**라고 부르고 있다. 하지만 이러한 용어들은 경우에 맞게 사용한다. 이 절의 뒤부분에서는 입출력모듈의 두가지 형태들을 모두 입출력통로라고 부르기로 한다.

직접기억기접근

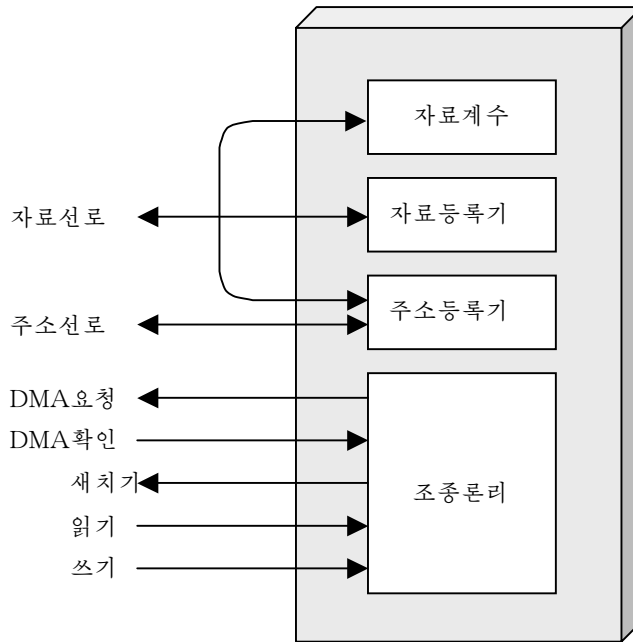



그림 11-2. 대표적인 DMA블로크도

그림 11-2에서는 일반적인 용어로 DMA론리라고 부르는것을 보여 주고 있다. DMA장치는 처리기로부터 체계의 조종을 인계 받고 처리기의 기능을 모방한다. 그것은 체계모선을 통하여 기억기의 안팎으로 자료를 이송한다. 대표적으로 DMA모듈은 처리기가 모선을 필요로 하지 않을 때에만 모선을 사용해야 한다. 그렇지 않으면 처리기가 일시 조작을 중단하도록 해야 한다. 이 수법은 보다 일반화된 방법으로서 DMA가 사실상 모선주기를 빼앗는것으로 되므로 주기빼앗기라고 부른다.

DMA방법은 다음과 같이 작업한다. 처리기가 자료블로크를 읽거나 쓰려고 할 때 다음과 같은 정보를 DMA모듈에 보내는 방법으로 그것에 지령을 발행한다. 즉

- 읽기 또는 쓰기가 요청될 때마다 처리기와 DMA모듈사이의 조종선로를 사용하여 요청되는 읽기 또는 쓰기의 조작형태
- 자료선로들에서 통신되는 관련된 입출력장치의 주소
- 자료모선을 통하여 통신되여 DMA모듈로 그것의 주소등록기에 보관되는 읽거나 쓰려는 기억기의 시작주소
- 자료모선을 통하여 다시 통신되여 자료계수등록기에 보관되는 읽거나 써넣으려는 단어수

처리기는 한편 다른 작업을 집행한다. 처리기는 DMA모듈에 입출력조작을 위임한다. DMA모듈은 처리기를 경유하지 않고 기억기의 안팎으로 직접 한개의 단어씩 자료블로크전체를 이송한다. 이송이 완료되면 DMA모듈은 처리기에 새치기신호를 보낸다. 따라서 처리기는 이송의 시작과 끝에서만 관여한다(그림 1-19 ㄷ).

그림 11-3에서는 처리기가 명령주기에서 중단되는 점을 보여 주고 있다. 매개 경우에 처리기는 모선사용을 요구하기직전에 중단한다. 그때 DMA장치는 한개의 단어를 이송하고 처리기에 조종을 넘긴다. 이것은 새치기가 아니다. 이때에는 처리기가 문맥을 보관하지 않으며 그 어떤 다른 작업을 한다. 오히려 처리기는 한개의 모선주기동안 림시정지한다. 총체적으로 보면 처리기가 보다 느리게 집행된다. 결국 다중단어의 입출력이송에는 DMA가 새치기구동식이나 프로그람식입출력보다 더욱 효과적이다.

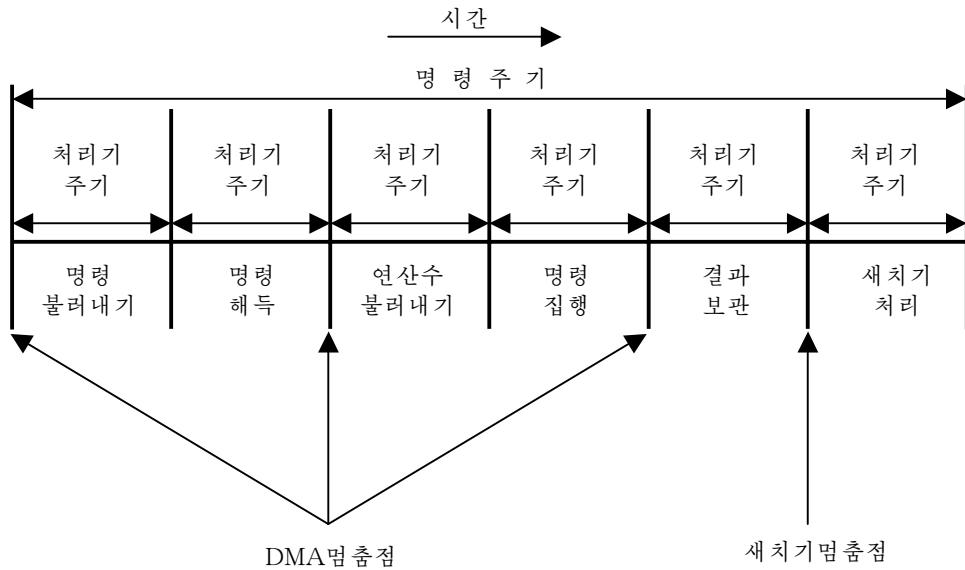



그림 11-3. 명령주기동안 DMA 및 새치기멈춤점

DMA기구는 여러가지 방식으로 구성할수 있다. 몇가지 구성가능성을 그림 11-4에 보여 주고 있다. 첫번째 실례에서는 모든 모듈들이 동일한 체계모선을 공유하고 있다. 처리기를 대리하여 동작하는 DMA모듈은 기억기와 입출력사이에서 자료를 교환하는데 프로그람식입출력방법을 사용한다. 이러한 구성에서는 비용이 많이 들지는 않지만 그대신 명백히 효률이 낮다. 처리기에 의해 조종되는 프로그람식입출력에서는 매개 단어이송에 두개의 모선주기(이송요청과 뒤이은 이송)가 소비된다.

필요한 모선주기수는 DMA가 입출력기능들을 통합함으로써 근본적으로 줄일수 있다. 그림 11-4 ㄴ에서 보여 주는바와 같이 이러한 구성에서는 DMA모듈과 체계모선을 포함하지 않는 한개 또는 그이상의 입출력모듈사이를 련결하는 경로가 있다. DMA론리는 실제상 입출력모듈의 한 부분으로 되거나 한개 또는 그이상의 입출력모듈들을 조종하는 개별적인 모듈로 될수 있다. 이러한 개념을 입출력모선을 사용하는 DMA모듈에 입출력모듈을 련결하는 방식으로 한단계 더 확장할수 있다(그림 11-4 ㄷ). 이것은 DMA모듈에서 입출력대면부의 수를 한개로 줄이고 구성을 쉽게 확장할수 있게 한다. 이 모든 경

우(그림 11-4 ㄴ와 ㄷ)에 DMA모듈이 처리기 및 주기억기와 공유하는 체계모선은 기억기와 자료를 교환하고 처리기와 조종신호들을 교환하기 위하여서만 DMA모듈에 의하여 사용된다. DMA와 입출력모듈사이에서의 자료교환은 체계모선밖에서 진행된다.

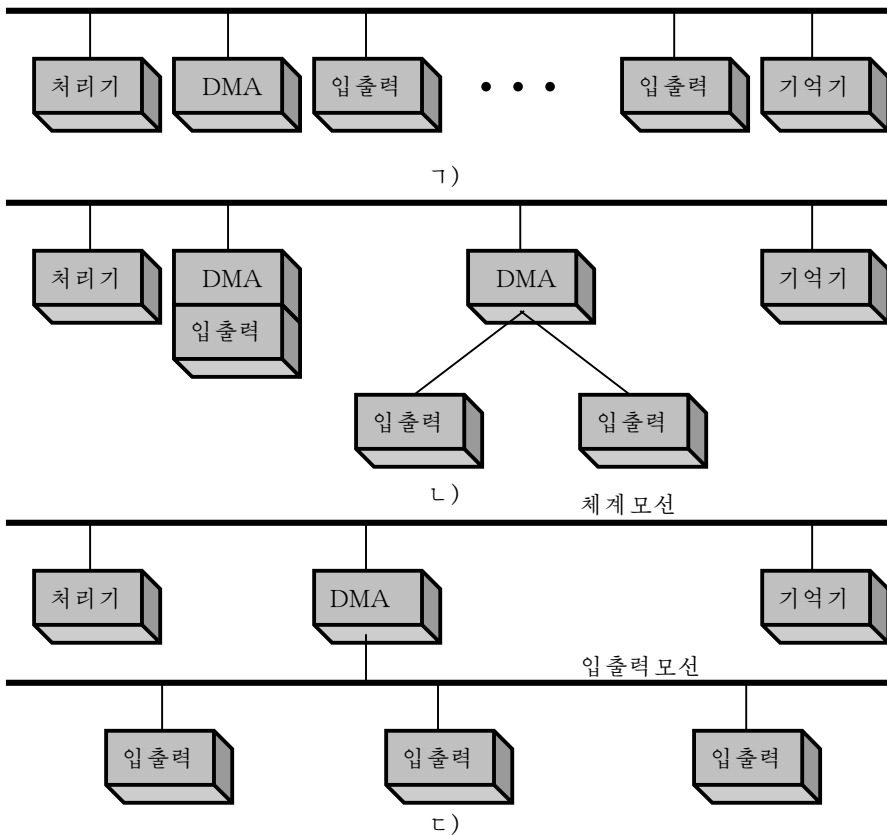



그림 11-4. 각이한 DMA 구성
ㄱ- DMA를 분리한 단일모선형, ㄴ-DMA입출력을 통합한 단일모형,
ㄷ- 입출력모선형

제 3 절. 조작체계의 설계문제

입출력설비를 설계하는데서 가장 중요한 두가지 목표는 효률과 일반성이다. **효률**은 입출력조작들이 흔히 계산체계에서 병목을 형성하므로 중요하다. 그림 11-1을 다시 보면 대부분의 입출력장치들이 주기억기나 처리기에 비하여 속도가 매우 뜨다는것을 알수 있다. 이 문제를 해결하는 한개 방도는 이미 본것처럼 프로쎄스가 집행되고 있는 동안 다른 프로쎄스들이 입출력조작들을 기다리도록 하는 다중프로그람처리방식을 실현하는것이다. 그러나 오늘날의 기계들에서 주기억기의 크기가 커짐에 따라 입출력이 처리기의 능력에 따라 서지 못하고 있다. 처리기의 차지를 유지하기 위하여 보충적인 준비프로쎄스를 끌어 들이기 위한 교체방법이 적용되고 있는데 이것은 자체내에서의 입출력조작이다. 이때 입출력설계에서 주요한 노력은 입출력의 효률을 개선하기 위한 방안을 세우는데 돌

려 진다. 디스크입출력은 그것의 중요성으로 하여 많은 주의가 돌려 지고 있고 이 장의 많은 부분에서 디스크입출력효률에 대한 설명을 하고 있다.

다른 중요한 목표는 **일반성**이다. 오유의 단순화와 오유로부터의 해방을 위하여 모든 장치들을 통일적인 방식으로 처리하는것이 필요하다. 이것은 프로쎄스들이 입출력장치들을 조사하는 방법과 조작체계가 입출력장치들과 조작들을 관리하는 방법에 적용한다. 장치의 특성지표들의 차이로 하여 실천적으로 일반성을 보장하기 어렵다. 될수 있으면 입출력기능의 설계에 계층적이면서 모듈적인 방법을 사용하여야 한다. 이 방법으로 낮은 준위의 루틴들에서 장치입출력의 대부분의 세부들을 은폐시켜줌으로써 사용자프로쎄스들과 조작체계의 웃준위들이 읽기, 쓰기, 열기, 닫기, 걸기, 걸기해제와 같은 일반적인 기능들로 장치들을 갈라 볼수 있다.

입출력기능의 론리적구조

제2장에서 체계구조에 대하여 보았지만 거기에서 벌써 현대적인 조작체계들의 계층적인 성질에 대하여 강조하였다. 계층의 원리는 조작체계의 기능들이 그것들의 복잡성, 특성시간척도, 그것들의 추상준위에 따라 달라 질수 있다는것이다. 이 방법에 의하면 조작체계의 조직을 여러개의 층으로 구성한다. 매개 층들은 조작체계의 필요한 기능들과 관련된 부분모임을 수행한다. 보다 기본적인 기능들을 수행하며 그러한 기능들의 세부들을 은폐시키는것은 보다 낮은 다음 층에 넘긴다. 그 층은 보다 높은 층에 봉사를 제공한다. 리상적으로는 층들이 한개의 층에서의 변화가 다른 층들에서의 변화를 필요로 하지 않도록 정의되여야 한다. 이로부터 한개의 문제를 다루기 쉬운 작은 문제들로 분해한다.

일반적으로 보다 낮은 층들은 훨씬 더 빠른 시간척도로 처리를 진행한다. 조작체계의 일부 부분들은 콤퓨터하드웨어와 직접 대화해야 하는데 이때 사건들은 몇십억분의 일초정도로 짧은 시간척도를 가진다. 스펙트르의 다른 끝에서 보면 조작체계의 부분들은 사용자와 통신을 진행하는데 사용자는 몇초에 한번씩인 정도의 훨씬 더 느린 속도로 지령을 발생한다. 층들의 모임을 사용하면 이러한 환경에 잘 어울릴수 있다.

입출력기능에 이러한 원리를 구체화하여 적용하면 그림 11-5에서 제안한 조직형태로 된다(표 2-4와 비교하라.). 이러한 조직의 세부들은 장치와 응용프로그람의 형태에 관계된다. 가장 중요한 세가지 론리구조들이 그림에 제시되여 있다. 물론 특정한 조작체계는 이러한 구조들과 정확히 일치하지 않을수 있다. 그러나 일반적인 원리들을 그대로 적용하고 있는 대부분의 조작체계들에서는 대체로 이 방법으로 입출력을 진행한다.

먼저 가장 간단한 경우로서 바이트렬이나 레코드들과 같은 간단한 방법으로 통신하는 국부주변장치에 대하여 고찰하자(그림 11-5 ㄱ). 이때 포함된 층들은 다음과 같다. 즉

- **론리입출력층 :** 론리입출력모듈은 장치를 론리자원으로 취급하며 실제적으로 장치를 조종하는 세부들과는 관계하지 않는다.
- **장치입출력층 :** 요청된 조작들과 자료(완충된 문자들, 레코드들과 기타)는 입출력명령들의 적당한 순서들, 통로지령들과 조종기지시들로 변환된다. 완충수법들은 사용률을 개선하는데 사용할수 있다.
- **일정작성 및 조종층 :** 입출력조작들의 실제적인 대기렬짓기와 일정작성은 이 층에서 진행하며 조작들에 대한 조종도 이 층에서 진행한다. 이때 새치기들은 이 층에서 처리하며 입출력상태가 수집되고 통보된다. 이것은 실제적으로 입출력모

듈을 따라서 장치하드웨어와 대화하는 쏘프트웨어층이다.

통신장치의 입출력구조(그림 11-5 ㄴ)는 우에서 설명한것과 꼭 같다. 기본차이는 론리적인 입출력모듈이 그자체가 많은 층들로 구성될수 있는 통신구성방식으로 바뀌여진것이다. 그러한 실례로서 부록 1에서 취급하는 TCP/IP를 들수 있다.

그림 11-5 ㄷ에서는 파일체계를 지원하는 2차기억기에서 입출력을 관리하기 위한 전형적인 구조를 보여 주고 있다. 아직 설명하지 않은 세개의 층들은 다음과 같은것들이다. 즉

- **등록부관리층 :** 이 층에서 기호들로 된 파일이름들은 직접적으로 또는 파일서술자나 색인표를 통하여 간접적으로 파일을 참조하는 식별자들로 변환된다. 이 층은 또한 추가, 삭제, 재조직 등과 같은 파일등록부에 작용하는 사용자조작들과 관련되여 있다.

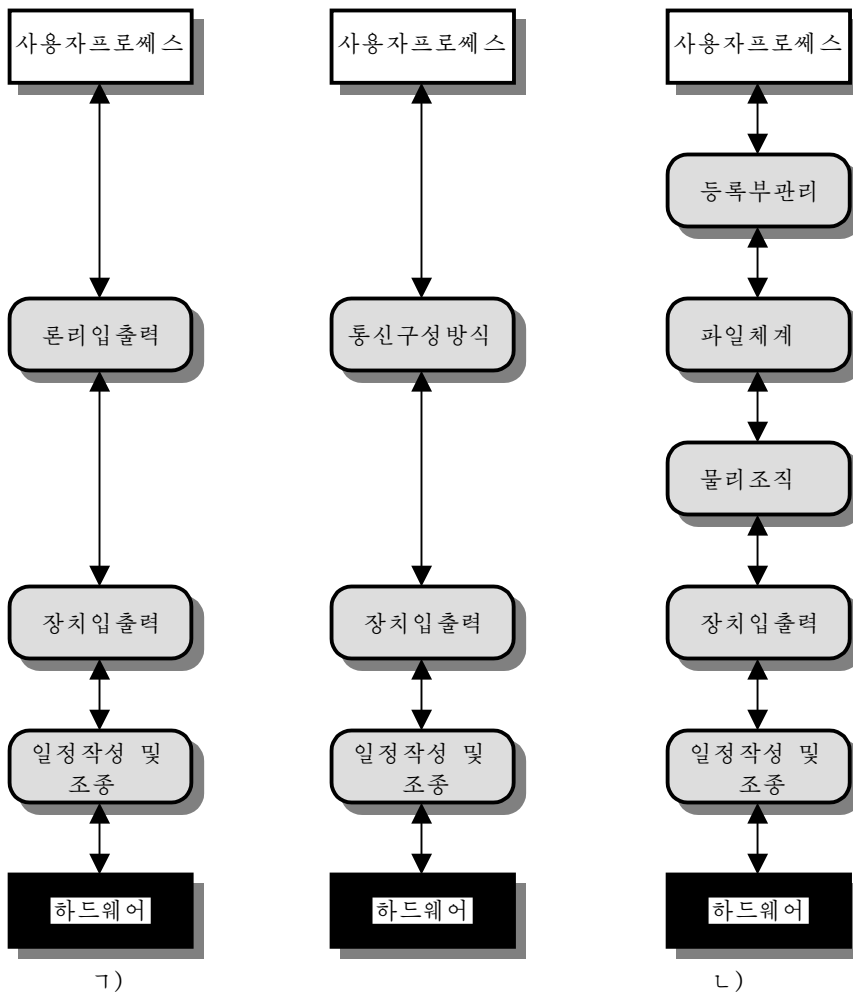



그림 11-5. 입출력조직의 모형
ㄱ-론리주변장치, ㄴ-통신포구, ㄷ-파일체계

- **파일체계층 :** 이 층에서는 파일들의 론리구조와 열기, 닫기, 읽기, 쓰기와 같은 사용자에 의해 명시될수 있는 조작들을 취급한다. 접근권은 또한 이 층에서 관리한다.
- **물리조직층 :** 가상기억기주소들은 토막화 및 페지화구조를 물리적인 주기억기주소들로 변환하여야 하며 파일들과 레코드들에 대한 론리적참조들은 2차기억장치의 물리적인 자리길과 분구, 물리적인 2차기억기주소들로 변환하여야 한다. 일반적으로 2차기억기의 공간과 주기억기의 완충기들의 배정도 역시 이 층에서 취급한다.

파일체계가 중요하기때문에 이 장과 다음장에서 그것의 각이한 구성요소들을 고찰하기로 한다. 이 장에서는 하위의 세개층에 주목하여 설명하고 제12장에서는 상위의 두개층에 대하여 취급한다.

제 4 절. 입출력의 완충조직

사용자프로쎄스가 매개 블로크의 길이가 512byte인 테프로부터 자료블로크들을 한번에 읽으려 한다고 하자. 가상위치 1000부터 1511까지 사용자프로쎄스의 주소공간의 자료령역으로 자료를 읽어 들이려 한다. 가장 간단한 방법은 테프장치에 대한 입출력지령(Read__Block[1000, tape]와 같은)을 집행하여 자료가 사용가능하게 되기를 기다리는것이다. 기다림은 차지기다림(장치상태를 부단히 검사한다.)이거나 보다 실제적으로는 새치기에 의한 프로쎄스중단일수 있다.

이 방법에는 두가지 문제가 있다. 우선 프로그람이 상대적으로 속도가 뜬 입출력이 완료되기를 기다리면서 잠시 정지하는것이다. 두번째 문제는 입출력에 대한 이 방법이 조작체계에 의한 교체결정과 충돌을 일으키는것이다. 1000～1511의 가상위치들은 블로크이송이 집행되는동안 주기억기에 유지되여야 한다. 한편 일부 자료는 소실될수 있다. 만일 페지화가 사용되고 있다면 적어도 목적위치들을 포함하는 페지는 주기억기에로 페쇄되여야 한다. 이때 프로쎄스의 일부분이 디스크에로 페지내기될수 있음에도 불구하고 이것이 조작체계에 의하여 필요하다고 해도 프로쎄스를 완전히 교체내기하는것은 불가능하다. 또한 단일프로쎄스교착의 위험성이 있다는것을 주의하자. 만일 프로쎄스가 입출력지령을 발행한다면 결과를 기다리면서 중단되고 조작이 시작되기전에 교체내기되며 프로쎄스가 입출력사건을 처리하면서 페색되고 입출력조작은 프로쎄스가 교체넣기되기를 기다린다. 이 교착을 피하기 위하여 입출력조작에 참가하는 사용자기억기는 입출력조작이 대기렬에 들어 서고 얼마동안 집행될수 없다고 하더라도 입출력요청이 발행된후에 즉시 주기억기에서 페쇄하여야 한다.

출력조작에 대해서도 고찰방법은 꼭 같다. 만일 블로크가 사용자프로쎄스령역으로부터 입출력모듈에로 직접 이송중에 있다면 프로쎄스는 이송되는동안 페색되고 프로쎄스는 교체내기되지 못할수 있다.

이러한 간접소비시간들과 비효률성들을 없애기 위하여 때로는 요청들이 되기전에 미리 입력이송들을 수행하고 요청이 된후에 얼마동안 출력이송들을 수행하는것이 편리하다. 이 수법을 완충이라고 한다. 이 절에서는 체계의 성능을 개선하기 위하여 조작체계들에 의하여 지원되는 완충방안 몇가지를 고찰한다.

각이한 완충방안들을 취급하는데서 때때로 두가지 형태의 입출력장치 즉 블로크지향형과 흐름지향형사이의 구별을 잘 해 두는것이 필요하다. **블로크지향**장치들은 보통 크기가 고정되여 있는 블로크들에 정보를 보관하며 이송들은 한번에 한개의 블로크씩 진행한다. 일반적으로 그것의 블로크번호에 의하여 자료를 참조할수 있다. 디스크들과 테프들은 블로크지향장치들의 실례이다. **흐름지향**장치들은 자료들을 블로크구조가 아닌 바이트흐름으로서 안팎으로 이송된다. 말단들, 인쇄기들, 통신포구들, 마우스 기타 지시장치들과 2차기억기가 아닌 대부분의 다른 장치들은 흐름지향형이다.

단일완충기

조작체계가 제공하는 지원의 가장 간단한 형태는 단일완충법이다(그림 11-6 ㄴ). 사용자프로쎄스가 입출력요청을 발행할 때 조작체계는 주기억기의 체계부분에 있는 완충기를 조작에 할당한다.

블로크지향장치들에서 단일완충방안은 다음과 같이 설명할수 있다. 즉 입력이송들은 체계의 완충기에로 진행된다. 이송이 완료되였을 때 프로쎄스는 블로크를 사용자공간으로 이동시키며 즉시 다른 블로크를 요청한다. 이것을 미리읽기 또는 선행입력이라고 부르는데 이것은 블로크가 드디여 필요될것이라는 기대속에서 진행된다. 많은 형태의 계산에서는 자료가 보통 순차적으로 접근되기때문에 시간의 대부분이 론리적인 가정에 바쳐 진다. 처리순서의 끝에서만 블로크를 필요 없이 읽는다.

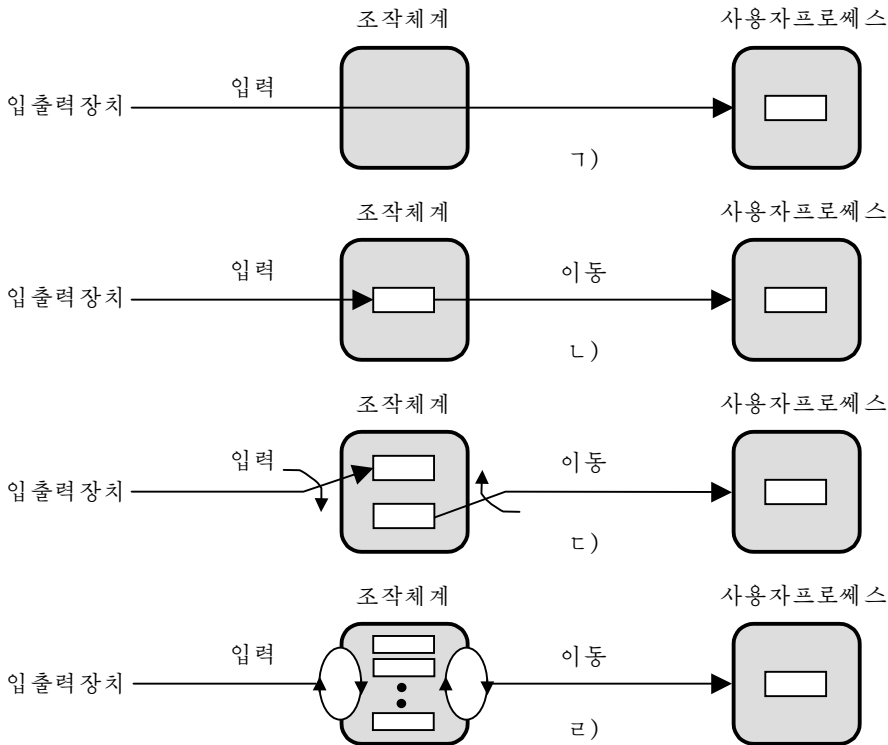



그림 11-6. 입출력완충방안(입력)
ㄱ-비완충법, ㄴ-단일완충법, ㄷ-2중완충법, ㄹ-순환완충법

이 방법은 일반적으로 체계의 완충이 없는 경우에 비하여 속도가 높다. 사용자프로쎄스는 다음의 블로크를 읽어 들이는 동안 한개의 자료블로크를 처리할수 있다. 조작체계는 입력조작이 사용자프로쎄스기억기가 아니라 체계기억기에로 진행되기때문에 프로쎄스를 교체내기할수 있다. 그러나 이 수법은 조작체계의 론리를 복잡하게 만든다. 조작체계는 사용자프로쎄스에 체계완충기들을 할당한 경로를 보존해야 한다. 교체론리도 역시 영향을 준다. 만일 입출력조작이 교체에 사용되는 동일한 디스크를 포함한다면 프로쎄스를 교체내기하기 위하여 동일한 장치에 디스크쓰기들을 대기렬짓는다는 느낌은 거의 들지 않는다. 프로쎄스를 교체하고 주기억기를 해방하려는 이러한 시도는 입출력조작이 끝날 때까지 하지 않는데 그 시간에 프로쎄스를 디스크에 교체하는것은 적합하지 않다.

블로크지향출력에 대해서도 우와 같이 고찰할수 있다. 자료가 장치에로 전송되고 있을 때 그것들은 먼저 사용자공간으로부터 체계완충기에로 복사되고 거기로부터 자료는 드디여 쓰기된다. 요청하고 있는 프로쎄스는 결국 자유상태에 있거나 필요할 때 교체된다.

[KNUT97]에서는 단일완충과 비완충사이의 완전하지는 못하지만 교육에 유익한 성능비교를 하고 있다. 한개의 블로크를 입력하는데 필요한 시간을 T, 입력요청들사이에 끼여 드는 계산시간을 C라고 하자. 완충이 없는 경우에 블로크당 집행시간은 $T + C$이다. 그런데 단일완충기인 경우에 그 시간은 $\max[C, T] + M$이다. 여기서 M은 자료를 체계완충기로부터 사용자기억기에로 옮기는데 필요한 시간이다. 대부분의 경우에 뒤의 값은 앞의 값보다 비할바 없이 작다.

흐름지향입출력에서 단일완충법은 단번행형식이나 단번바이트형식에 사용할수 있다. 단번행조작은 흘리기방식의 말단(때로는 벙어리말단이라고도 부른다.)들에 적합하다. 이러한 형식의 말단에서 사용자입력은 한번에 한개의 행씩 하는데 행의 끝에서 복귀신호를 낸다. 말단에로의 출력은 입력에서와 류사하게 한번에 한개의 행씩 진행한다. 행인쇄기는 그러한 장치에 대한 하나의 실례로 된다. 단번바이트조작형식은 매개 건반치기를 중요시하는 양식방식의 말단들과 수감기와 조종기들과 같은 다른 많은 주변장치들에 사용되고 있다.

단번행입출력에서 완충기는 한개의 행을 유지하는데 사용할수 있다. 사용자프로쎄스는 완전한 행이 도착하기를 기다리면서 입출력을 하는 동안 중단된다. 출력을 할 때 사용자프로쎄스는 출력행을 완충기에 배치하고 처리를 계속할수 있다. 이때 프로쎄스처리는 완충기가 첫번째 출력조작에 의해 비기전에 두번째 출력행을 보내지 않는이상에는 중단할 필요가 없다. 단번바이트형식인 경우에 조작체계와 사용자프로쎄스사이의 대화는 제5장에서 취급한 생산자/소비자모형에 따른다.

2중완충기

입출력조작에 두개의 체계완충기를 할당함으로써 단일완충법을 개선하였다(그림 11-6 ㄷ). 프로쎄스가 한개의 완충기에로(로부터) 자료를 이송하는 사이에 조작체계는 다른 완충기를 비운다(채운다). 이 수법을 **2중완충법** 또는 **완충기교체법**이라고 한다.

블로크지향이송에서 $\max[C, T]$로서 집행시간을 대략 짐작할수 있다. 이때 $C \leq T$이면 블로크지향장치가 완전한 속도로 계속 진행할 가능성이 있다. 한편 $C > T$이면 2중완충법에 의해 프로쎄스가 확실히 입출력에 관여하지 않아도 된다. 어느 경우에도 단일완충법에 비하여 특성이 개선된다. 또한 이 개선에서는 복잡성이 증가된것만큼 가격이 증가된다.

흐름지향입력에서는 다시 두가지 선택적인 조작방식에 직면하게 된다. 단번행입출력형식에서 사용자프로쎄스는 그것이 2중완충기에 앞서 실행되지 않는한 입력 및 출력을 중단할 필요가 없다. 단번바이트조작에서 2중완충기는 길이가 두배인 단일완충기에 비하여 특정한 우점을 발휘하지 못한다. 두 경우 모두 생산자/소비자모형에 따른다.

순환완충기

2중완충기방안에서는 입출력장치와 프로쎄스사이에서 자료의 흐름을 균일하게 내보내야 한다. 만일 특정한 프로쎄스의 성능이 관심속에 주목된다면 입출력조작이 프로쎄스와 함께 유지되도록 해야 한다. 만일 프로쎄스가 고속의 입출력무리들을 수행한다면 2중완충법은 적합하지 않다. 이경우에는 두개이상의 완충기들을 사용해야 문제를 해결할수 있다.

두개이상의 완충기들을 사용할 때 완충기들의 집합을 순환완충기(그림 11-6 ㄹ)라고 부르는데 여기서 매개 개별적인 완충기는 순환완충기의 한개 단위로 된다. 이것은 단순히 제5장에서 취급한 경계완충기의 생산자/소비자모형이다.

완충법의 편의프로그람

완충법은 입출력요구의 첨두값들을 고루롭게 하여 주는 수법이다. 그러나 프로쎄스의 평균요구가 입출력장치가 봉사할수 있는것보다 클 때에는 완충량이 입출력장치로 하여금 프로쎄스와 명확히 보조를 맞출수 있게 한다. 완충기가 여러개 있다고 하여도 모든 완충기들이 결국 가득차게 되고 프로쎄스는 자료의 매개 토막을 처리한후에 기다려야 한다. 그러나 다중프로그람처리환경에서 입출력작업이 각이하고 봉사하려는 프로쎄스작업이 각이할 때 완충법은 조작체계의 효률과 프로쎄스들의 성능을 높일수 있는 한가지 수법으로 된다.

제 5 절. 디스크일정작성

지난 30년동안 처리기들과 주기억기의 속도증가는 디스크접근속도의 증가를 훨씬 릉가하였는데 처리기와 주기억기의 속도는 두자리 증가한 반면에 디스크의 속도는 한자리 증가하였다. 결국 디스크의 속도는 현재 적어도 주기억기에 비하여 네자리정도 떨어지고 있다. 이 간격은 앞으로도 계속 유지될것으로 보고 있다. 그리하여 디스크기억부분체계의 성능은 매우 중요한 관심사로 되고 있고 그것의 성능을 개선하기 위한 많은 연구가 진행되여 왔다. 이 절에서는 그와 관련된 일부 중요한 문제들을 강조하면서 가장 중요한 방법들을 고찰한다. 디스크체계의 성능이 파일체계설계문제들과 밀접히 관련되므로 제12장에서 계속 취급한다.

디스크성능파라메터

디스크입출력조작의 실제적인 세부는 콤퓨터체계와 조작체계, 입출력통로의 특성과 디스크조종기의 하드웨어에 관계된다. 디스크입출력이송의 박자동기선도를 그림 11-7에 제시하였다.

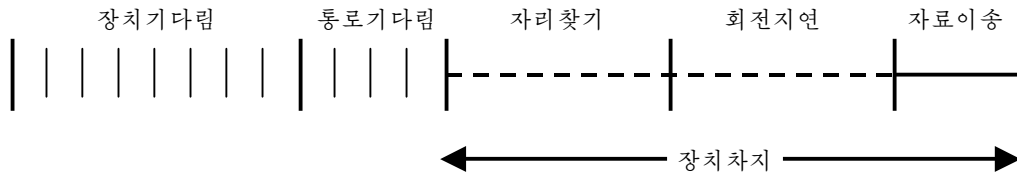



그림 11-7. 디스크입출력이송의 박자동기선도

디스크구동기가 동작할 때 디스크는 일정한 속도로 회전한다. 읽기나 쓰기를 위하여 자두는 해당한 자리길을 차지하고 자리길의 해당한 분구의 시작점을 차지해야 한다 .[1] 자리길을 선택하자면 가동자두체계에서는 자두를 이동시켜야 하고 고정자두체계에서는 한개의 자두를 정기적으로 선택하여야 한다. 가동자두체계에서 자리길에 자두가 자리잡는 시간을 **자리찾기시간**이라고 한다. 자리길이 일단 선택되면 디스크조종기는 적당한 분구가 자두와 줄을 맞추도록 회전할 때까지 기다려야 한다. 자두를 분구의 시작점에 도달시키는데 걸리는 시간을 **회전지연시간** 또는 회전기다림시간이라고 한다. 자리찾기시간과 회전지연시간의 합을 **접근시간**이라고 하는데 이것은 읽거나 쓰려는 위치를 차지하는데 걸리는 시간이다. 자두가 일단 자리를 차지하면 읽기나 쓰기조작은 분구가 자두밑을 이동할 때 진행된다. 이것은 자료이송조작의 일부분이다. 접근시간과 이송시간외에 디스크입출력조작과 정규적으로 관련된 몇가지 대기렬지연들이 있다. 프로쎄스가 입출력요청들을 발행할 때 그것은 우선 장치를 사용가능하게 하는 대기렬에서 기다려야 한다. 그 시간에 장치는 프로쎄스에 할당된다.

만일 장치가 한개의 입출력통로와 입출력통로들의 모임을 다른 디스크구동기들과 공유한다면 통로를 사용가능하게 하기 위해 보충적으로 기다릴수 있다. 그러한 시점에서 디스크접근을 시작하기 위하여 자리찾기를 진행한다.

일부 대형콤퓨터체계들에서는 회전위치수감(RPS)이라고 부르는 수법을 사용하고 있다. 이 작업은 다음과 같이 한다. 자리찾기지령이 발행되면 통로는 다른 입출력조작을 처리하기 위하여 해방된다. 자리찾기가 완료되면 장치는 언제 자료가 자두밑으로 올것인가를 결정한다. 그 분구가 자두에 접근할 때 장치는 거꾸로 주콤퓨터에로 가는 통신선로를 도로 설정하기 위한 시도를 한다. 만일 조종장치나 통로가 다른 입출력에 의해 차지되고 있다면 재련결은 실패하고 장치는 다시 련결을 시도하기전에 옹근 한 회전을 순환해야 하는데 이것을 RPS실패라고 한다. 이것은 그림 11-7의 시간선상에 첨가되여야 하는 여분의 지연요소로 된다.

자리찾기시간

자리찾기시간은 디스크팔을 요구하는 자리길에 이동시키는데 필요한 시간이다. 이것은 일정한 시간으로 보장하기 곤난한 량이라는것을 알수 있다. 자리찾기시간은 두가지 기본요소 즉 초기시동시간과 그리고 일단 접근팔이 속도를 내여 횡단해야 할 자리길들을 가로 지르는데 걸리는 시간으로 되여 있다. 공교롭게도 가로지름시간은 자리길수에 대한

---

[1] 디스크조직과 양식화를 취급한 부록 11-ㄱ를 보시오.

선형함수로 되지 않고 시동시간과 설정시간(자두가 목적자리길우에 자리 잡은 후에 자리길식별이 확정될 때까지의 시간)을 포함한다.

디스크요소들은 보다 작게, 보다 가볍게 함으로써 커다란 개선을 가져 왔다. 몇년전에 대표적인 디스크는 직경이 14인치(36cm)였다면 오늘날 대부분의 디스크는 직경이 공통적으로 3.5인치(8.9cm)로 되여 팔이 이동해야 할 거리를 줄이였다.

회전지연시간

플로피디스크가 아닌 자기디스크들의 회전속도는 5400~10000rpm인데 이것은 웃한계에서 보면 6ms당 한 회전하는것과 맞먹는다. 따라서 10000rpm에서 평균회전지연시간은 3ms이다. 플로피디스크는 대표적으로 300~600rpm사이에서 회전한다. 이때 평균지연시간은 100ms~200ms사이에 있다.

이송시간

디스크와 오고가는 자료의 이송시간은 다음의 식에서와 같이 디스크의 회전속도에 관계된다. 즉

$$T = \frac{b}{rN}$$

여기서 T = 이송시간
b = 이송해야 할 바이트수
N = 자리길의 바이트수
r = 초당회전수로 표시된 회전속도

이때 총체적인 평균접근시간은 다음과 같이 표시할수 있다. 즉

$$T\alpha = Ts + \frac{1}{2r} + \frac{b}{rN}$$

여기서 Ts는 평균자리찾기시간이다.

박자동기의 비교

앞에서 정의한 파라메터들에 의해 평균값을 믿을 위험성을 보여 주는 두가지 서로 다른 입출력조작들을 살펴 보자. 광고한 평균자리찾기시간이 10ms이고 회전속도가 10000rpm, 자리길당 320분구이고 매개 분구가 512byte인 대표적인 디스크를 고찰하자. 총 1.3MB인 2560분구로 된 파일을 읽으려고 한다. 이때 총 이송시간을 대략 계산하여 보자.

먼저 파일이 디스크에서 가능한 치밀하게 기억되여 있다고 보자. 즉 파일은 8개의 린접한 자리길의 모든 분구들(8자리길×320분구/자리길=2560분구)을 차지한다. 이것을 순차조직이라고 한다. 이제 첫번째 자리길을 읽는 시간은 다음과 같다. 즉

평균자리찾기	10ms
회전지연	3ms

320분구읽기　　　　$\underline{6\text{ms}}$
　　　　　　　　　19ms

나머지 자리길들은 본질적으로 자리찾기시간이 없이 읽을수 있다고 가정하자. 즉 입출력조작은 디스크로부터 나가는 흐름을 따라 갈수 있다고 보자. 이때 련속된 매개 자리길은 3 + 6 = 9ms내에 읽어 진다. 완전한 파일을 읽기 위해서는 다음의 시간이 소비된다.

총 시간 = 19×7×9 = 82ms = 0.082s

이제 순차적인 접근이 아니라 자유접근방식으로 동일한 자료를 읽는데 필요한 시간을 계산하여 보자. 즉 분구들에 대한 접근률은 디스크에서 임의로 분산된다. 매개 분구에 대하여 다음의 시간이 걸린다.

평균자리찾기	10ms
회전지연	3ms
1분구의 읽기	0.01875ms
	13.01875ms

총 시간 = 2560×13.01875 = 33328ms = 33.328s

분구들을 디스크로부터 읽어 내는 순서는 입출력성능에 엄청난 효과를 낸다는것이 명백하다. 여러 분구들을 읽고쓰는 파일접근인 경우에 자료분구들을 산개시키는 방법에 대한 어떤 조종을 하는데 다음장에서 이 제목에 대한것을 지적한다. 그러나 파일접근인 경우조차 다중프로그람처리환경에서 동일한 디스크에 대하여 경쟁하는 입출력요청들이 있을수 있다. 이때 디스크입출력의 성능을 디스크에 대한 순전한 자유접근에 의해 달성하는것보다 더 좋게 개선할수 있는 방법들을 조사하는것이 유익하다.

디스크일정작성방책

방금 서술한 실례에서 성능의 차이를 가지게 되는 리유는 자리찾기시간에서 찾을수 있다. 만일 분구접근요청들이 자유로 자리길들의 선택을 동반한다면 디스크입출력의 성능은 현실적으로 빈약하다. 문제를 개선하기 위하여서는 자리찾기들에 소비되는 평균시간을 줄이는것이 필요하다.

조작체계가 매개 입출력장치에 대한 요청대기렬을 유지하는 다중프로그람처리환경에서 대표적인 정황을 고찰하자. 단일디스크에서 대기렬의 각이한 프로쎄스들로부터 많은 입출력요청들(읽기와 쓰기)이 있을수 있다. 만일 대기렬에서 자유순서로 항목들을 선택한다면 방문하려는 자리길들이 자유로 발생하여 나쁜 성능을 주게 될것이라는것을 예상할수 있다. 이 **확률적일정작성**방법은 다른 수법들을 평가하기 위한 견본으로서 쓸모가 있다.

가장 단순한 일정작성법은 **선입선출**(FIFO)일정작성법인데 이것은 대기렬의 항목들을 단순히 순차적으로 처리한다는것을 의미한다. 이 전략은 공평한것이 우점인데 그것은 매개 요청이 봉사를 받되 접수된 순서로 봉사를 받기때문이다. 그림 11-8 ㄱ에서는 FIFO법으로 디스크팔을 이동시키는 경우를 보여 주고 있다. 이 실례에서는 디스크의 자리길수가 2000이고 디스크요청대기렬이 자유요청들에 의한것이라고 가정하였다.요청된 자리길들을 접수한 순서는 55, 58, 39, 18, 90, 160, 150, 38, 184이다. 표 11-2 ㄱ에서

는 그 결과를 보여 주고 있다. FIFO법에서 만일 접근을 요구하는 프로쎄스들이 몇개밖에 없다면 그리고 많은 요청들이 파일분구들을 클라스터화하려고 한다면 좋은 성능을 기대할수 있다. 그러나 이 수법은 디스크에 대하여 경쟁하는 많은 프로쎄스들이 있는 경우에 흔히 성능에서 확률적일정작성법에 가깝다. 이것은 보다 정교한 일정작성전략을 고찰

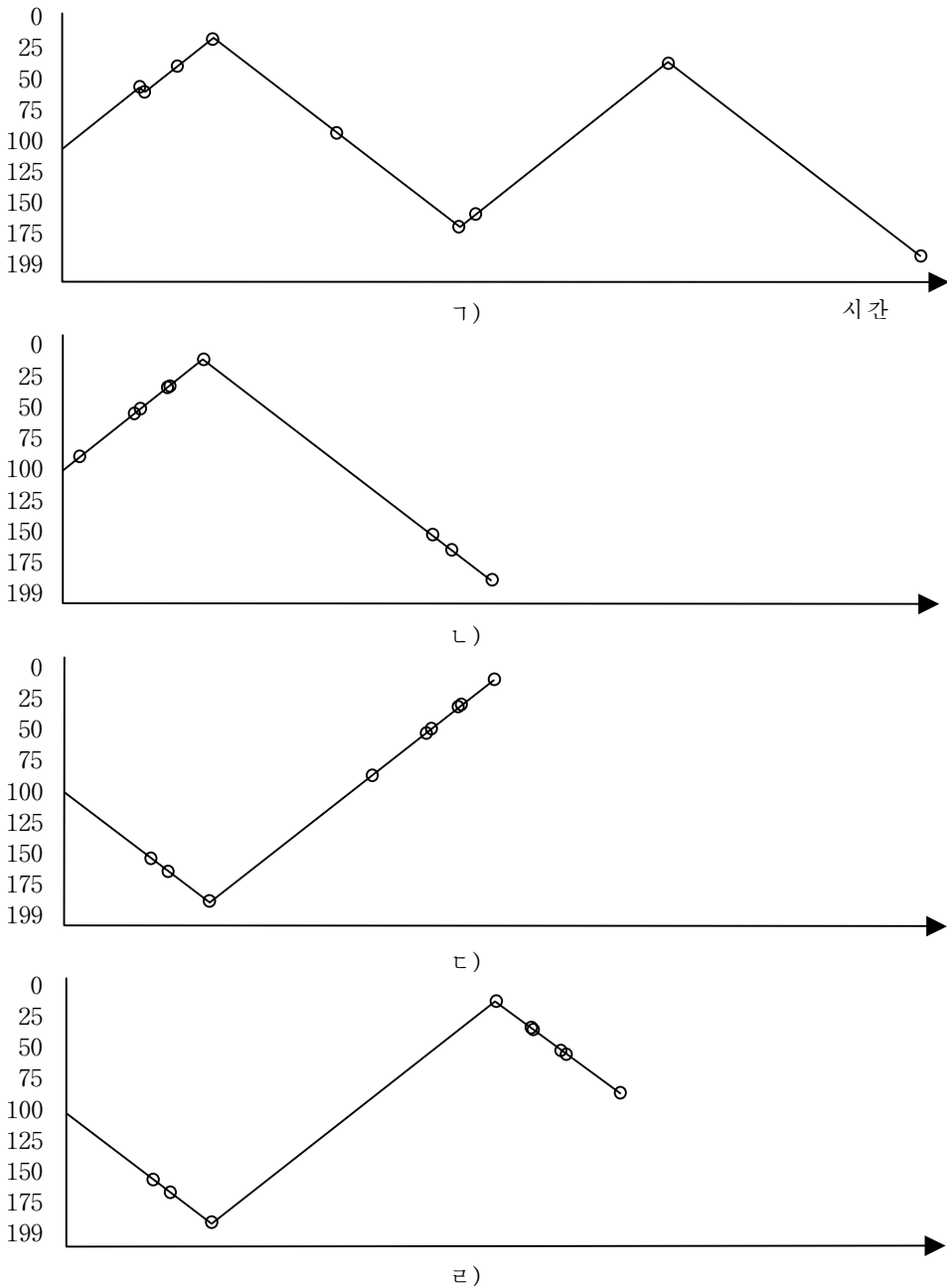



그림 11-8. 디스크일정작성알고리듬의 비교(표 11-3을 보시오.)
ㄱ-FIFO, ㄴ-SSTF, ㄷ-SCAN, ㄹ-C-SCAN

하는데 유익할수 있다. 이 많은것들을 표 11-3에 목록화하여 제시하였는데 이제부터 구체적으로 고찰하여 보자.

표 11-2. 디스크일정작성알고리듬

ㄱ) FIFO (자리길 100에서 시작)		ㄴ) SSTF (자리길 100에서 시작)		ㄷ) SCAN (자리길 100에서 자리길번호가 증가하는 방향으로 시작)		ㄹ) S-SCAN (자리길 100에서 자리길번호가 증가하는 방향으로 시작)	
다음의 접근자리길	**가로 지르는 자리길수**	**다음의 접근자리길**	**가로 지르는 자리길수**	**다음의 접근자리길**	**가로 지르는 자리길수**	**다음의 접근자리길**	**가로 지르는 자리길수**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
평균자리찾기길이	55.3	**평균자리찾기길이**	27.5	**평균자리찾기길이**	27.8	**평균자리찾기길이**	35.8

우선권

우선권에 기초한 체계(PRI)에서 일정작성의 조종은 디스크관리쏘프트웨어의 조종밖에서 진행된다. 이러한 방법은 디스크의 사용률을 최적화하기 위한것이 아니라 조작체계안의 다른 목적을 실현하기 위한것이다. 흔히 짧은 일괄일감들과 대화일감들에는 장시간의 계산을 필요로 하는 보다 긴 일감들보다 보다 높은 우선권이 부여된다. 이것은 많은 짧은 일감들이 체계와 즉시에 조화되고 좋은 대화응답시간을 보장할수 있다. 그러나 보다 긴 일감들은 과도하게 긴 시간동안 기다릴수 있다. 그러한 방책에 대하여 일부 사용자들이 대응책을 세울수 있는데 그들은 체계를 촉진시키기 위하여 일감들을 보다 작은 쪼각들로 분할한다.

후입선출법

가장 최근의 요청들을 항상 택하는 전략은 일정한 우점을 가진다. 트랜잭션처리체계에서 가장 최근의 사용자에게 장치를 배당하면 순차적인 파일을 따라 팔을 조금 이동시키거나 전혀 이동시키지 않을수 있다. 이러한 국소성의 원리를 사용하면 처리능력을 개선할수 있고 대기렬의 길이를 줄일수 있다. 일감이 실제적으로 파일체계를 사용하고 있는 동안에 그것은 가능한 고속으로 처리된다. 그러나 만일 디스크가 큰 작업부하로 인하여 차지상태를 유지한다면 서로 다른 고갈이 발생할수 있는 가능성이 있다. 일단 일감이 입출력요청을 대기렬에 넣고 선로의 자두로부터 물러 났다면 일감은 그것앞의 대기렬이 비지 않는이상 선로의 자두를 복귀할수 없다.

표 11-3. 디스크일정작성알고리듬[WIED87]

이름	설명	주의
	요청자에 따르는 선택	
RSS	확률적일정작성법	분석과 모의를 위한것이다.
FIFO	선입선출법	모든것들중에서 가장 공평하다.
PRI	프로쎄스우선권법	디스크대기렬관리범위를 넘어 조종한다.
LIFO	후입선출법	국소성과 자원사용률을 최대화한다.
	요청항목에 따르는 선택	
SSTF	최단봉사시간우선법	사용률이 높고 대기렬이 작다.
SCAN	디스크정역방향법	봉사분산이 더 좋다.
CSCAN	고속복귀형SCAN법	봉사가변성이 낮다.
N걸음 SCAN	한번에 N개의 레코드를 주사하는 N걸음 SCAN법	봉사를 보증한다.
FSCAN	SCAN주기의 시작점에서 대기렬의 크기가 N인 N걸음 SCNA법	적재를 수감한다.

FIFO, 우선권, LIFO(후입선출)일정작성법은 다만 대기렬이나 요청자의 할당들에 기초하고 있다. 만일 현재의 자리길위치가 일정작성자에게 알려 져 있다면 요청된 항목에 기초한 일정작성법을 사용할수 있다. 이 방책들을 다음에 고찰하자.

최단봉사시간우선법

최단봉사시간우선(CSTF)방책은 디스크의 팔을 현재의 위치로부터 가장 적게 이동시킬것을 요구하는 디스크입출력요청을 선택하는것이다. 따라서 자리찾기시간이 최소로 되도록 선택한다. 물론 최소자리찾기시간을 선택하면 항상 많은 팔이 운동들에 대한 평균자리찾기시간이 최소로 된다는 담보는 없다. 그러나 이것은 FIFO법보다 더 좋은 성능을 보장한다. 팔이 두 방향으로 이동할수 있기때문에 자유련결차단알고리듬은 거리가 같은 경우들을 해결하는데 사용할수 있다.

그림 11-8 ㄴ와 표 11-2 ㄴ에서는 FIFO법을 사용한 동일한 실례에 대한 SSTF법의 성능을 보여 주고 있다.

SCAN법

FIFO법을 제외하고 지금까지 설명한 모든 방책들에서는 수행되지 못한 어떤 요청을 대기렬이 완전히 빌 때까지 처리하지 못할수 있다. 즉 현존요청에 앞서 선택하게 될 새로운 요청들이 항상 도착할수 있다. 이 짧은 시간동안의 고갈을 방지하는 단순한 방법이 바로 SCAN알고리듬이다.

SCAN에서 팔은 오직 한 방향으로만 이동하기 위하여 필요한데 미해결중에 있는 모든 요청들은 이동도중에 만족시키면서 이동방향에서 마지막자리길에 도달할 때까지 또는 이동방향에서 더는 요청을 하지 않을 때까지 이동한다. 이 마지막 경우를 때로는 LOOK방책이라고도 한다. 다음에 봉사방향은 거꾸로 되여 주사는 반대방향으로 진행되며 다시 모든 요청들을 순서대로 접수한다.

그림 11-8 ㄷ와 표 11-2 ㄷ에서는 SCAN방책을 설명해 주고 있다. 보는바와 같이 SCAN방책은 SSTF방책과 거의 동일하게 동작한다. 사실 팔이 실례의 시작점에서 자리길번호가 작아 지는 방향으로 이동하고 있다고 가정하였다면 일정작성패턴은 SSTF법이나 SCAN법과 동일하다. 그러나 이것은 새로운 항목들이 대기렬에 첨가되지 않는 정적인 실례이다. 대기렬이 동적으로 변화되고 있는 경우에도 SCAN법은 요청패턴이 별다른 것을 제외하고는 SSTF법에서와 류사하다.

SCAN방책은 가장 최근에 가로 지른 구역과 반대쪽으로 편위된다. 이것은 SSTF나 지어 LIFO와 같이 국소성을 사용하지 않는다.

SCAN방책은 일감들의 요청들이 제일 안쪽 또는 제일 바깥쪽의 자리길들에 제일 가까운 자리길들에 대한것이라면 그러한 일감들에 보다 유리하고 또한 제일 마감에 도착하는 일감들에도 유리하다는것을 쉽게 알수 있다. 첫번째 문제는 CSCAN법에 의하여 피할수 있고 두번째 문제는 N걸음 SCAN법에 의하여 극복할수 있다.

CSCAN법

CSCAN(순환식 SCAN)법은 한 방향으로만 주사하도록 제한한다. 한 방향으로 전진하여 마지막 자리길을 방문하였을 때 팔은 디스크의 반대끝으로 되돌아 오면서 주사를 다시 시작한다. 이것은 새로운 요청들에 의하여 부닥치는 최대지연을 감소시킨다. SCAN법에서 만일 안쪽 자리길로부터 바깥쪽 자리길로 주사하는데 걸리는 예상시간을 t라고 하면 원주의 분구들에 대하여 예상되는 봉사시간간격은 $2t$이다. 그러나 CSCAN법에서는 그 간격이 $t+s_{max}$정도로 된다. 여기서 s_{max}는 최대자리찾기시간이다. 그림 11-8 ㄹ에서는 CSCAN법의 동작을 보여 주고 있다.

N걸음 SCAN법 및 FSCAN법

SSTF, SCAN 및 CSCAN법들에서는 고찰하는 시간주기동안 팔을 이동시키지 않아도 될수 있다. 실례로 한개 또는 몇개의 프로쎄스들이 한개의 자리길에 대한 고속접근을 한다면 그것들은 그 자리길에 대한 요청들을 반복함으로써 장치를 완전히 독점할수 있다. 고밀도다중면디스크들은 저밀도디스크들이나 한개 또는 두개의 면을 가진 디스크들보다 이 특성지표들에 의하여 더 큰 영향을 받을수 있다. 이 《팔의 점착성》을 회피하기 위하여 디스크요청대기렬을 토막화하는데 여기서 토막은 한번에 완전히 처리할수 있는것이다. 이 방법의 두가지 실례가 바로 N걸음 SCAN법과 FSCAN법이다.

N걸음 SCAN법에서는 디스크요청대기렬을 길이가 N인 부분대기렬로 토막화한다. 부분대기렬들은 SCAN법을 사용하여 한번에 한개씩 처리한다. 대기렬을 처리하고 있는 동안 새로운 요청들은 다른 대기렬에 첨가하여야 한다. 만일 이보다 적은 요청들이 주사의 끝에서 사용가능하다면 그것들모두는 다음의 주사와 동시에 처리된다. N의 값이 클 때 N걸음 SCAN법의 성능은 SCAN법에 가깝다. N=1일 때에는 FIFO법을 채용한다.

FSCAN법은 두개의 부분대기렬을 사용하는 방책이다. 주사가 시작될 때 모든 요청들은 대기렬중의 한개에 들어 있고 다른 대기렬은 비여 있다. 주사를 진행하는 동안 모든 새로운 요청들은 다른 대기렬에 들어 간다. 이때 새로운 요청들에 대한 봉사는 낡은 요청들이 처리될 때까지 연기된다.

제 6 절. RAID

앞에서 설명한바와 같이 2차기억기의 성능개선률은 처리기들이나 주기억기의 개선률보다 현저히 낮다. 이러한 불일치는 디스크기억체계가 총체적인 콤퓨터체계성능을 개선하는데서 기본초점으로 된다.

콤퓨터성능의 다른 령역에서와 같이 디스크기억기설계자들은 지금까지는 한개의 요소를 꽂아 넣었다면 이제는 다중병렬요소들을 사용하여 보충적인 성능리득을 얻어야 한다는것을 인식하게 되였다. 디스크기억기인 경우에 이러한 인식은 독립적으로 그리고 병렬로 동작하는 디스크배렬을 개발하는데로 떠밀었다. 다중디스크에서 개별적인 입출력요청들은 필요한 자료가 개별적인 디스크들에 상주하는 동안 병렬로 처리할수 있다. 더우기 단일한 입출력요청은 접근하려는 자료블로크가 다중디스크들에 걸쳐 분산되여 있는 경우에 병렬로 집행할수 있다.

다중디스크들을 사용할 때 자료를 조직하고 믿음성을 개선하기 위하여 여분을 첨가할수 있는 방도에는 여러가지가 있다. 이것은 많은 가동환경들과 조작체계들에 사용할수 있는 자료기지방안들을 개발하는것을 곤난하게 할수 있다. 다행히도 RAID(독립디스크여분배렬)라고 부르는 다중디스크자료기지의 설계에 대한 표준방안이 공업적으로 실현되였다. RAID방안은 0준위～6준위로 된 7개의 준위[2]로 이루어 져 있다. 이 준위들은 계층적인 관계를 의미하지 않지만 세가지 공통적인 특성지표들을 공유하는 각이한 설계방식을 지적한다. 즉

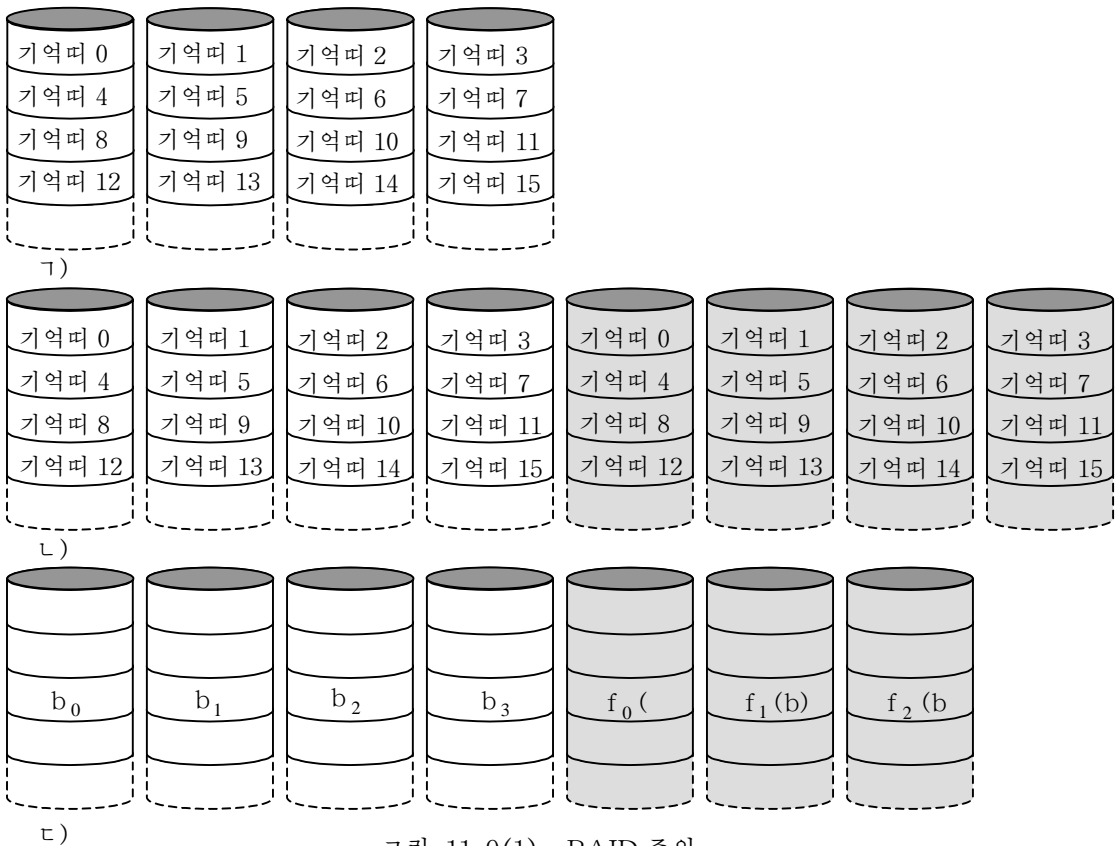



그림 11-9(1). RAID 준위
ㄱ-RAID 0(비여분), ㄴ-RAID 1(대칭복제), ㄷ-RAID 2(하밍코드에 의한 여분)

---

[2] 일부 연구사들과 회사들에 의하여 보충적인 준위들이 정의되기는 하였으나 이 절에서 설명하는 7개의 준위들은 일반적으로 쓰이는것들이다.

1. RAID는 조작체계가 한개의 론리구동기로 보는 물리적인 디스크구동기모임이다.
2. 자료는 배렬의 물리적인 구동기들에 걸쳐 분산된다.
3. 여분디스크용량은 디스크고장시 자료의 회복가능성을 담보하여 주는 기우성정보를 보관하는데 사용된다.

두번째 및 세번째 특성지표들의 세부들은 각이한 RAID준위에 따라 차이난다. RAID 0은 세번째 특성지표를 지원하지 않는다.

표 11-4. RAID준위

부류	준위	설명	입출력요청률 (읽기/쓰기)	자료이송속도 (읽기/쓰기)	대표적인 응용들
기억띠가르기형	0	비여분	큰 기억띠: 우수하다.	작은 기억띠: 우수하다.	비림계자료에 대한 높은 성능을 요구하는 응용들
대칭복제형	1	대칭복제	좋다/공평하다	적당하다/공평하다	체계구동기들:림계파일들
병렬접근형	2	하밍코드	나쁘다	우수하다	
	3	비트기우성	나쁘다	우수하다	도형, CAD와 같은 대규모입출력요청크기의 응용들
독립접근형	4	블로크기우성	우수하다/공평하다	적당하다/나쁘다	
	5	블로크교차식분산	우수하다/공평하다	적당하다/나쁘다	높은 요청률, 읽기집중, 자료찾아보기
	6	블로크교차식2중분산	우수하다/나쁘다	적당하다/나쁘다	극히 높은 사용성을 요구하는 응용률

RAID라는 용어는 본래 Berkeley에 있는 캘리포니아대학의 연구사그룹이 론문에서 처음으로 쓴것이다[PATT88].[3] 론문에서는 각이한 RAID구성들과 응용들의 틀거리들을 세웠으며 현재 사용하고 있는 RAID준위들을 정의하였다. RAID전략은 대용량의 디스크구동기들을 여러개의 소용량의 구동기들로 교체하고 다중구동기들로부터 자료의 동시접근을 할수 있게 하는 방법으로 자료를 분산시킴으로써 입출력성능을 개선하고 보다 쉽게 용량을 증가시킨다.

RAID제안이 유일하게 기여한것은 여분의 필요성을 유효하게 지적한것이다. 다중자두들과 조절기들을 동시에 동작시킴으로써 보다 높은 입출력 및 이송속도를 보장할수 있게 되였지만 다중구동기들의 사용으로 하여 고장확률이 증가된다. 떨어 지는 믿음성을 보상하기 위하여 RAID에서는 디스크고장으로 인하여 소실된 자료를 회복할수 있게 하

---

[3] 론문에서 RAID는 단어결합 저가격디스크여분배렬의 첫 문자들을 붙여 만든것이다. 여기서 저가격이라는 말은 RAID배렬의 상대적으로 소형인 값눅은 디스크들을 다른것 즉 단일한 대형의 고가격디스크(SLED)와 대조하기 위하여 사용하였다. SLED는 본질상 RAID와 비RAID구성을 위해 사용하고 있는 기술과 류사한 기술을 적용한 과거의 제품이다. 따라서 공업에서는 RAID배렬이 높은 성능과 믿음성을 보장한다는것을 강조하기 위하여 독립이라는 말을 채용하였다.

는 이미 기억된 기우성정보를 사용한다. 이제 매개 RAID준위들을 보기로 하자. 표 11-4에서는 7개의 준위들을 요약하여 제시하였다. 이것들중에서 2준위와 4준위는 상업적으로 제공되지 않았고 공업적으로 실현하는데 적당하지 못하다. 그럼에도 불구하고 이 준위들에 대한 서술은 일부 다른 준위들에서 설계선택들을 명백히 하는데 도움을 준다.

그림 11-9의 실례들에서는 여분이 없는 4개의 디스크들을 필요로 하는 자료용량을 지원하기 위한 7개의 RAID방안들을 사용하는 경우들을 보여 주고 있다. 그림에서 사용자자료와 여분의 자료의 틀거리를 강조하고 각이한 준위들의 상대적인 기억요구사항들을 보여 주고 있다. 이 그림을 참조하면서 다음의 설명을 해나가도록 한다.

RAID 0준위

RAID 0준위는 성능을 개선하기 위한 여분을 포함하고 있지 않기때문에 성능과 용량이 주요관심사로 되고 저가격이 믿음성개선보다 더 중요한 초고속콤퓨터들에서 일부 응용되고 있다.

RAID 0에서 사용자 및 체계자료는 배렬의 모든 디스크들에 걸쳐 분산된다. 이것은 단일한 대형디스크를 사용할 때보다 커다란 우점을 가진다. 만일 두개의 서로 다른 입출력요청들이 두개의 서로 다른 자료블로크들에 대한것이라면 요청된 블로크들이 서로 다른 디스크들에 있을수 있다. 그러면 두개의 요청들은 병렬로 발행될수 있고 따라서 입출력대기렬시간을 줄일수 있다.

그러나 다른 모든 RAID준위들에서와 같이 RAID 0에서도 디스크배렬에 걸쳐 자료를 단순한 방법으로 분산시키지는 않는다. 즉 자료는 사용가능한 디스크들에 걸쳐 기억띠화한다. 이것은 그림 11-10을 보면 잘 알수 있다. 모든 사용자 및 체계자료는 하나의 론리디스크에 기억되여 있는것처럼 보인다. 디스크는 기억띠들로 분할한다. 기억띠들로서는 물리적인 블로크들, 분구들 또는 다른 단위들이 될수 있다. 기억띠들은 련속적인 배렬성원들에 순환식으로 사영된다. 한개의 기억띠가 매개 배렬성원에로 정확히 사영되는 론리적으로 련속되는 기억띠들의 모임을 띠무늬라고 부른다. n디스크배렬에서 첫 n개의 론리기억띠들은 첫 띠무늬를 형성하는 n개의 디스크의 매개 디스크에 첫번째 기억띠로서 물리적으로 기억된다. 두번째 n개의 기억띠들은 매개 디스크의 두번째 기억띠들로 분산된다. 이런 식으로 계속되여 나간다. 이러한 구성의 우점은 한개의 입출력요청이 여러개의 론리적으로 련속인 기억띠들로 이루어 질 때 그 요청에 대한 n개까지의 기억띠들을 병렬로 처리할수 있고 따라서 입출력이송시간을 단축할수 있는것이다.

그림 10-10에서는 론리디스크공간과 물리디스크공간사이에서 사영하기 위한 배렬관리쏘프트웨어의 사용정형을 보여 주고 있다. 쏘프트웨어는 디스크나 주콤퓨터에서 집행할수 있다.

고속자료이송용량용 RAID 0

임의의 RAID준위들의 성능은 주체계의 요청패턴과 자료의 배치상태에 크게 관계된다. 이 문제는 RAID 0에서 가장 명백하게 지적할수 있는데 여기서는 여분도의 영향이 분석에 지장을 주지 않는다. 먼저 RAID를 사용하여 자료이송속도를 높이는 문제를 고찰하자. 응용에서 높은 이송속도를 얻기 위해서는 두가지 요구사항을 만족해야 한다. 우선 고속이송능력은 주기억기와 개별적인 디스크구동기들사이의 온전한 경로에 관계된다.

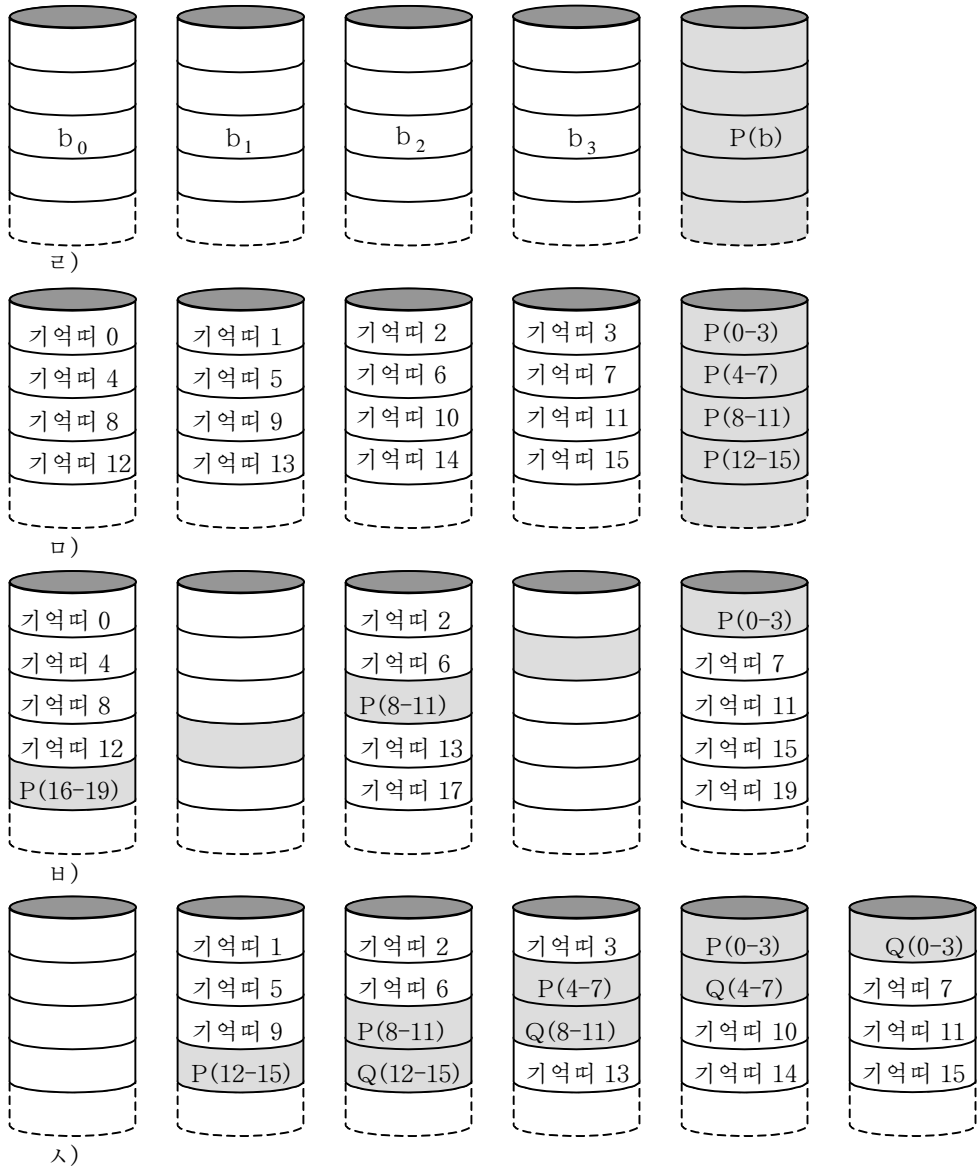



그림 11-9(2). RAID준위(계속)
ㄹ-RAID 3(비트교차기우성), ㅁ-RAID 4(블로크준위기우성)
ㅂ-RAID 5(블로크준위분산기우성), ㅅ-RAID 6(2중여분)

이것은 내부조종기의 모선들, 주체계의 입출력모선들, 입출력접속기들과 주기억기모선들을 포함한다.

두번째 요구는 응용프로그람에서 디스크배렬을 효률적으로 구동할수 있게 입출력요청들을 내보내도록 하는것이다. 이 요구는 대표적인 요청이 기억띠의 크기와 비교할만한 많은 량의 론리적으로 련속된 자료에 대한것이라면 자연히 만족된다. 이 경우에 한개의 입출력요청이 여러개의 디스크로부터 자료의 병렬이송을 동반하므로 단일디스크이송에 비하여 유효이송속도가 증가된다.

고입출력요청률용 RAID 0

트랜잭션지향환경에서 사용자는 이송속도보다 응답시간에 더 관심을 가진다. 적은 량의 자료에 대한 개별적인 입출력요청인 경우에 입출력시간은 디스크자두의 이동(탐색시간)과 디스크의 회전(회전기다림시간)에 좌우된다.

트랜잭션환경에서는 초당 수백개의 입출력요청들이 있을수 있다. 디스크배렬은 다중디스크들에서 입출력부하의 균형을 맞추면서 높은 입출력집행속도를 보장한다. 유효부하평균은 대체로 다중입출력요청들이 현저히 많을 때에만 이루어 진다. 이것은 다중독립응용들이나 다중비동기입출력요청들을 할수 있는 단일트랜잭션지향의 응용이 가능하다는 것을 의미한다. 성능은 또한 기억띠의 크기에 따라 좌우된다. 만일 단일입출력요청이 단일디스크접근을 동반하도록 기억띠의 크기가 상대적으로 크다면 입출력을 기다리는 다중요청들을 병렬로 처리하여 매개 요청에 대한 대기렬시간을 줄일수 있다.

RAID 1준위

RAID 1은 여분도를 달성하는 방법에서 2준위～6준위들과 서로 차이난다. 다른 RAID방안들에서는 여분을 조성하기 위하여 일부 형태의 기우성계산법을 사용하는데 RAID 1에서는 모든 자료를 2중화하는 단순한 방책으로 여분을 조성한다. 그림 11-9 ㄴ에서 보는바와 같이 RAID 0에서처럼 자료의 기억띠화방법을 사용한다. 그러나 이 경우에 매개 론리기억띠는 두개의 개별적인 물리디스크에로 사영되며 배렬의 매개 디스크가 동일한 자료를 보관하는 대칭복제디스크를 가지도록 한다.

RAID1의 조직에서는 여러가지 좋은 점들이 있다. 즉

1. 읽기요청은 요청된 자료를 가지고 있는 두개의 디스크중 어느 하나에 의하여 봉사 받을수 있는데 어떤 디스크이든 최소의 탐색시간 + 회전기다림시간을 보장한다.
2. 쓰기요청은 대응하는 두개의 기억띠를 갱신할것을 요구하는데 이것은 병렬로 진행할수 있다. 이때 쓰기작업은 두개의 쓰기동작중의 느린것(즉 탐색시간 + 회전기다림시간이 큰것)을 기준으로 지령을 준다. 그러나 RAID 1과의 《쓰기벌칙》은 없다. RAID의 2준위～6준위에서는 기우성비트들을 사용한다. 따라서 한개의 기억띠를 갱신할 때 배렬관리쏘프트웨어는 문제의 실제적인 기억띠를 갱신하는것과 함께 기우성비트를 먼저 계산하고 갱신하여야 한다.
3. 고장회복이 간단하다. 구동기가 고장나면 자료는 두번째 구동기로부터 여전히 호출할수 있다.

RAID 1의 기본결함은 가격이다. 이것은 그것을 지원하는 론리디스크의 두배의 디스크공간을 요구한다. 그것때문에 RAID 1의 구성은 체계쏘프트웨어와 자료와 그밖의 중요한 파일들을 보관하는 구동기들로 제한된다. 이러한 경우들에 RAID 1은 디스크고장시 중요한 모든 자료를 여전히 즉시에 사용할수 있도록 모든 자료의 실시간여벌을 제공한다.

트랜잭션지향의 환경에서 RAID 1은 요청무리의 대부분이 읽기일 때 높은 입출력요청률을 보장할수 있다. 이때 RAID 1의 성능은 RAID 0의 2배에 가까와 갈수 있다. 그러나 입출력요청들의 절대다수가 쓰기요청이라면 RAID 0에 비하여 뚜렷한 성능리득을 얻을수 없다. RAID 1은 또한 읽기비률이 높은 자료이송집중형의 응용들에서 RAID 0에

비하여 성능개선을 보장할수 있다. 만일 응용프로그람에서 매개 읽기요청을 두개의 디스크성원들이 관여하도록 분할할수 있는 경우에는 성능개선을 가져 올수 있다.

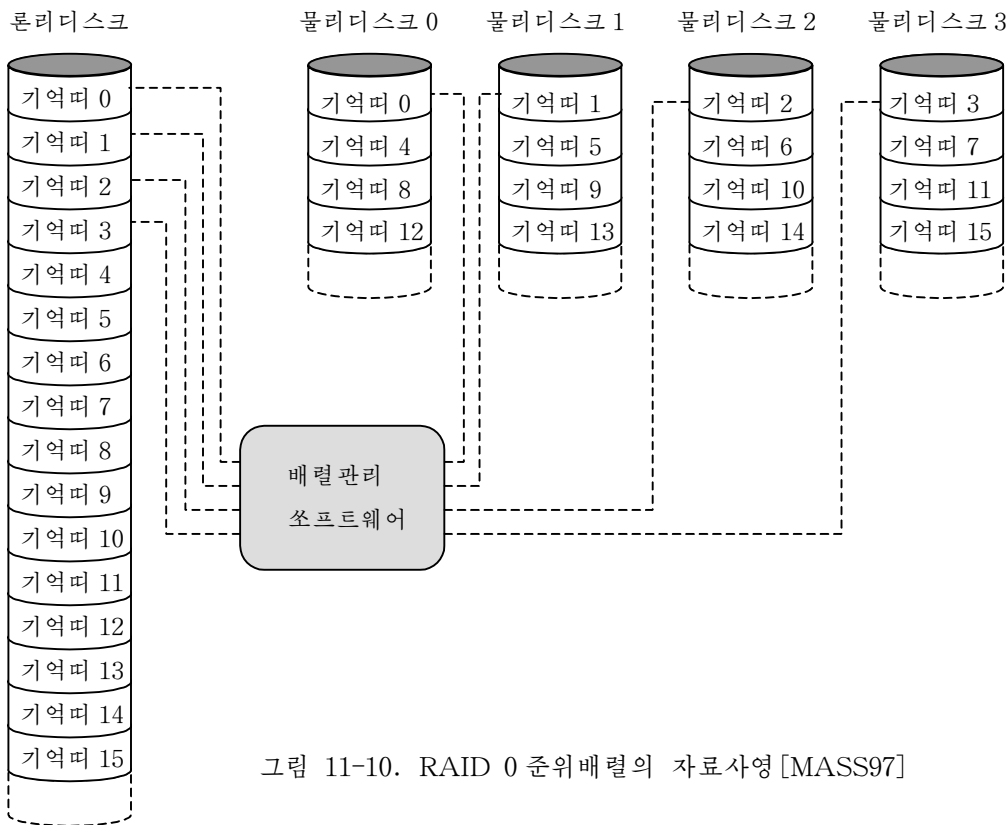



그림 11-10. RAID 0준위배렬의 자료사영[MASS97]

RAID 2준위

RAID의 2준위와 3준위에서는 병렬접근수법을 사용하고 있다. 병렬접근배렬에서 모든 성원의 디스크들은 매개 입출력요청의 집행에 참가한다. 대표적으로 개별적인 구동기들의 주축들은 매개 디스크머리가 임의의 순간에 매개 디스크의 동일한 위치에 놓이도록 동기화된다.

다른 RAID방안들에서와 같이 자료의 분해법을 사용한다. RAID 2와 3의 경우에 기억띠들은 보통 한개의 바이트 또는 단어와 같이 대단히 작다. RAID 2에서 오유정정코드는 매개 디스크의 대응하는 비트들에 걸쳐 계산되며 코드의 비트률은 다중기우성디스크들의 대응하는 비트위치들에 기억된다. 대표적으로 한개 비트의 오유를 수정하고 두개 비트의 오유를 발견할수 있는 하밍코드를 사용한다.

RAID 2가 RAID 1보다 더 적은 개수의 디스크를 요구함에도 불구하고 아직도 가격이 비싸다. 여분디스크개수의 로그값에 비례한다. 단일읽기조작에서 모든 디스크들이 동시에 호출된다. 요청된 자료와 그것과 관련된 오유정정코드가 배렬조종기에로 전송된다. 만일 한개의 비트가 오유이면 조종기는 읽기접근시간이 늦어 지지 않도록 오유를 즉시에

인식하고 수정할수 있다. 단일쓰기조작에서 모든 자료디스크와 기우성디스크들은 쓰기조작을 위해서 접근되여야 한다.

RAID 2는 많은 디스크오유가 발생하는 환경에서 효과적으로 선택사용할수 있다. 개별적인 디스크들과 디스크구동기들의 믿음성이 높으면 RAID 2는 그리 필요 없고 실현하지도 않는다.

RAID 3준위

RAID 3은 RAID 2와 류사한 방식으로 조직되여 있다. 차이점은 RAID 3에서는 디스크배렬의 규모에는 관계 없이 한개의 여분디스크만이 요구된다는것이다. RAID 3은 작은 기억띠들로 분산된 자료들에 대하여 병렬접근을 진행한다. 오유정정코드대신에 모든 자료디스크들의 동일한 위치의 개별비트모임에 대하여 단순한 기우성비트를 계산한다.

여분도

구동기고장사건이 일어 났을 때 기우성구동기를 호출하며 자료는 나머지 장치들로부터 재구축된다. 일단 고장난 구동기만을 교체하면 실패된 자료는 새로운 구동기에서 회복할수 있고 조작을 다시 시작할수 있다.

자료의 복구는 매우 단순하다. $X0$～$X3$디스크는 자료를, $X4$는 기우성디스크인 5개의 구동기를 가진 배렬을 고찰하자. 이때 i번째 비트에 대한 기우성은 다음과 같이 계산된다.

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

이제 X1이 고장이라고 하자. 웃식의 량변에 $X4(i) \oplus X1(i)$를 더한다면 다음과 같이 된다.

$$X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$$

이때 $X1$자료의 매개 기억띠의 내용들은 배렬의 나머지 디스크들의 대응한 기억띠들의 내용으로부터 재발생된다. 이 원리는 RAID의 3준위～6준위에서도 성립한다.

디스크고장사건이 발생한 경우에도 모든 자료는 축소방식으로 여전히 사용할수 있다. 이 방식에서 읽기들인 경우에 실패한 자료는 안맞음론리합계산을 통하여 재발생된다. 자료가 축소된 RAID 3배렬에 씌여질 때 기우성의 일관성이 재발생후에도 보장되여야 한다. 동작을 완전히 회복하자면 고장난 디스크를 교체하고 고장난 디스크의 모든 내용을 새로운 디스크에 재발생시켜야 한다.

성능

자료가 대단히 작은 기억띠들은 분할되여 있기때문에 RAID 3은 매우 좋은 자료이송속도를 보장할수 있다. 임의의 입출력요청은 모든 자료디스크로부터 자료의 병렬전송을 동반한다. 한편 한번에 한개의 입출력요청만을 집행할수 있다. 이때 트랜잭션지향환경에서 성능은 떨어 진다.

RAID 4준위

RAID의 4준위～6준위에서는 독립적인 접근수법을 사용한다. 독립적인 접근배렬에서는 매개 성원디스크가 개별적인 입출력요청들을 병렬로 만족시킬수 있도록 독립적으로 동작한다. 그러므로 독립적인 접근배렬들은 높은 입출력요청률을 요구하는 응용들에는

보다 적당하지만 높은 자료이송속도를 요구하는 응용들에는 상대적으로 그리 적합하지 않다.

다른 RAID방안들에서와 같이 여기서도 자료의 띠분할법을 사용한다. RAID의 4준위~6준위인 경우에 기억띠는 상대적으로 크다. RAID 4에서는 매개 자료디스크의 대응한 기억띠들에 대하여 비트별기우성기억띠를 계산하며 기우성비트들은 기우성디스크의 대응하는 기억띠에 기억시킨다.

RAID 4에서는 규모가 작은 입출력쓰기요청들이 수행될 때 쓰기벌칙을 동반한다. 쓰기가 발생할 때마다 매번 배렬관리쏘프트웨어는 사용자자료뿐아니라 대응하는 기우성비트들을 갱신하여야 한다. X0~X3은 자료디스크이고 X4는 기우성디스크인 5개의 구동기로 된 배렬을 고찰하자. 디스크 X1의 기억띠를 포함하는 쓰기가 진행된다고 하자. 초기에 매개 비트 i에 대하여 다음의 관계가 성립한다. 즉

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

갱신된후에 변경된 비트들을 빗선기호로 표시하면

$$\begin{aligned} X4'(i) &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) = \\ &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \oplus X1(i) \oplus X1(i) \\ &= X4(i) \oplus X1(i) \oplus X1'(i) \end{aligned}$$

새로운 기우성을 계산하기 위하여 배렬관리쏘프트웨어는 낡은 사용자기억띠와 낡은 기우성기억띠를 읽어야 한다. 다음에 그것은 두개의 기억띠들을 새로운 자료와 새롭게 계산된 기우성으로 계산하여야 한다. 이때 매개 기억띠의 쓰기는 두번의 읽기와 두번의 쓰기를 동반한다.

입출력쓰기가 모든 디스크구동기들의 기억띠를 포함하는 규모가 큰 경우에 기우성은 새로운 자료비트들만을 사용하여 계산하는 방법으로 쉽게 계산된다. 이때 기우성구동기는 자료구동기들과 병렬로 갱신될수 있으며 여분의 읽기들이나 쓰기들은 없다.

어떤 경우에든지 매개 쓰기조작은 기우성디스크를 포함하여야 하는데 따라서 이것은 병목현상을 일으킬수 있다.

RAID 5준위

RAID 5는 RAID 4와 류사한 방식으로 조직된다. 차이점은 RAID 5에서는 기우성기억띠들을 모든 디스크들에로 분산시킨다는것이다. 이때 대표적인 배정법은 순환법인데 그것을 그림 11-9 ㅂ에서 보여 주고 있다. n디스크배렬에서는 기우성기억띠가 첫 n개의 기억띠에 대하여 각이한 디스크에 배치되는데 이러한 패턴이 계속 반복된다.

기우성기억띠들을 모든 구동기에 분산시킴으로써 RAID 4에서와 같은 단일기우성디스크의 잠재적인 입출력병목현상을 피할수 있다.

RAID 6준위

RAID 6은 Berkeley의 연구자들에 의하여 다음의 론문에 도입되였다[KATZ89]. RAID 6방안에서는 두개의 서로 다른 기우성계산을 진행하고 각이한 디스크들의 개별적인 블로크들에 기억한다. 사용자의 자료가 N개의 디스크를 요구하는 RAID 6배렬은 N+2개의 디스크들로 구성된다.

그림 11-9 ㅅ에서는 이 방안을 보여 주고 있다. P와 Q는 두개의 서로 다른 자료검사알고리듬이다. 그중 한개는 RAID 4와 RAID 5에서 사용한 안맞음론리합계산이다. 다른것은 독립적인 자료검사알고리듬이다. 이 알고리듬에 의하여 두개의 디스크에 실패할 사용자자료가 들어 있다고 해도 자료를 재발생시킬수 있다.

RAID 6의 우점은 그것이 극히 높은 자료사용가능성을 보장한다는것이다. 세개의 디스크는 자료가 사용불가능하게 하는 MTTR(평균보수시간)간격이내에 고장이 나야 한다. 한편 RAID 6은 매개 쓰기가 두개의 기우성블로크에 작용하므로 실제적인 쓰기벌칙을 초래한다.

제 7 절. 디스크캐쉬

제1장의 제6절과 부록 1-ㄱ에서는 이미 캐쉬기억기의 원리들을 요약하였다. 용어 캐쉬기억기는 보통 주기억기보다 용량은 더 작고 속도는 더 빠르며 주기억기와 처리기사이에 위치한 기억기를 가리키는데 쓰인다. 그러한 캐쉬기억기는 국소성의 원리를 사용하여 평균기억접근시간을 줄인다.

그러한 원리를 디스크기억기에도 적용할수 있다. 명확히 말하면 디스크캐쉬는 주기억기에 있는 디스크분구용완충기이다. 캐쉬에는 디스크의 어떤 분구의 사본을 보관한다. 특정한 분구에 대한 입출력요청이 있을 때에는 분구의 내용이 디스크캐쉬에 있는가를 확정하기 위한 검사를 진행한다. 만일 있다면 요청은 캐쉬를 통해 만족된다. 그러나 만일 없다면 요청된 분구는 디스크로부터 디스크캐쉬에로 읽어 낸다. 참조의 국소성으로 인하여 한개의 입출력요청을 만족시키기 위하여 자료블로크를 캐쉬에로 불러 낼 때 앞으로 그것과 동일한 블로크를 참조할 가능성이 크다.

설계고찰

여기에 흥미 있는 몇가지 설계문제가 있다. 첫번째 설계문제는 입출력일정이 디스크캐쉬에 의해 만족될 때 디스크캐쉬의 자료를 요청중인 프로쎄스에로 발송해야 한다. 이것은 주기억기의 자료블로크를 디스크캐쉬로부터 사용자프로쎄스에 할당된 기억기에로 이송하거나 단순하게는 공유기억기능을 사용하여 지시자를 디스크캐쉬의 해당한 홈 에로 넘겨 주는 방법으로 집행할수 있다. 뒤의 방법은 기억기 대 기억기이송시간을 절약하며 또한 제5장에서 서술한 읽기자/쓰기자모형을 사용하여 다른 프로쎄스들로 공유된 접근을 할수 있다.

두번째 설계문제는 치환전략과 관련한것이다. 새로운 분구를 디스크캐쉬에로 끌어들일 때 현존하는 블로크들의 내용은 교체해야 한다. 이것이 제8장에서 서술한 일치문제이다. 거기서 제기된 요구사항은 페지치환알고리듬에 대한것이였다. 많은 알고리듬들이 제기되였다. 가장 일반적으로 사용된 알고리듬은 최대미사용(LRU)법이다. 여기서는 캐쉬안에서 가장 오래동안 참조하지 않은 블로크를 교체한다. 론리적으로 캐쉬는 블로크들의 탄창으로 되여 있는데 가장 최근에 참조된 블로크를 탄창의 꼭대기에 놓는다. 캐쉬의 블로크가 참조되였을 때 그것은 탄창의 현존위치로부터 탄창의 꼭대기에로 이동한다. 블로크를 2차기억기에로 끌어 넣을 때에는 탄창의 밑바닥에 있는 블로크를 제거하고 들어 오는 블로크를 탄창의 꼭대기우에 밀어 넣는다. 물론 이 블로크들을 주기억기안에서 여기저기에로 이동시키는것은 실제적으로 필요 없다. 이때 지시자의 탄창을 캐쉬와 련관시킬수 있다.

다른 알고리듬으로는 **최소빈도사용**(LFU)법이 있다. 여기서는 가장 적게 참조된 모임의 블로크를 교체한다. 블로크를 끌어 들일 때 그것은 계수값 1을 할당한다. 그리고 그 블로크를 참조할 때마다 매번 그 계수값은 1씩 증가된다. 치환이 요구될 때 계수값이

가장 작은 블로크가 선택된다. 직관적으로 LFU법이 선택프로쎄스의 매개 블로크에 대한 보다 적합한 정보를 사용할수 있게 하므로 LFU법이 LRU법보다 더 합리적인 방법이라고 볼수 있다.

단순한 LFU알고리듬은 다음의 문제를 제기한다. 전체적으로 볼 때 어떤 블로크들은 상대적으로 드물게 참조될수 있지만 그것들이 참조될 때 국소성에 의하여 반복되는 참조들의 간격이 작아 지는 경우들이 있을수 있고 따라서 높은 참조계수값을 얻을수 있다. 그러한 간격이 끝나면 참조계수기값은 흐트러 질수 있고 따라서 블로크가 곧 다시 참조될것이라는 확률을 반영하지 못한다.이로부터 국소성의 효과는 실제적으로 LFU알고리듬으로 하여금 치환선택들을 나쁘게 한다.

LFU의 이러한 난문제를 극복하기 위하여 [ROBI90]에서는 빈도에 기초한 치환법이라고 부르는 수법을 제기하였다. 명백히 하기 위하여 먼저 그림 11-11 ㄱ에 보여 주고 있는 단순화된 변종을 고찰하자. 블로크들은 론리적으로 LRU알고리듬과 같이 탄창으로 조직되여 있다. 탄창의 꼭대기의 어떤 부분은 새로운 구간으로서 제기된다. 캐쉬명중이 있을 때 참조된 블로크는 탄창의 꼭대기에로 이동한다. 만일 블로크가 이미 새로운 구간에 있었다면 그것의 참조개수값은 증가되지 않는다. 그렇지 않은 경우에는 그것이 1씩 증가된다. 충분히 큰 새로운 구간이 주어 졌다면 이것은 짧은 시간간격내에 반복하여 재참조되고 있는 블로크들에 대한 참조계수값들을 변화시킴이 없이 그대로 유지하도록 한다. 실패하면 새로운 구간에 있지 않으면서 가장 작은 참조계수값을 가진 블로크가 치환을 거쳐 선택된다. 이때 최근에 가장 적게 사용된 블로크가 결국 선택된다.

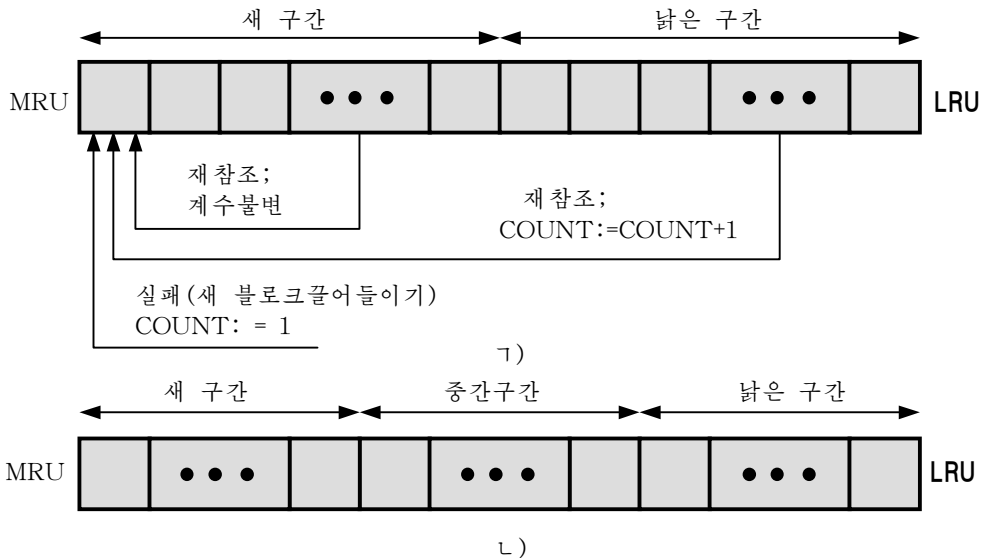



그림 11-11. 빈도에 의한 치환방법
ㄱ-FIFO, ㄴ-자유구간의 사용

연구자들은 이 방법이 LRU법에 비하여 성능이 조금밖에 개선된것이 없다고 보고
연구자들은 이 방법이 LRU법에 비하여 성능이 조금밖에 개선된것이 없다고 보고하고 있다. 문제점은 다음과 같다. 즉

1. 캐쉬실패인 경우에 새로운 블로크는 계수값을 1로 하고 새로운 구간에로 끌어들인다.
2. 계수값은 블로크가 새로운 구간에 남아 있는 동안 1로 유지된다.

3. 결국 블로크는 새로운 구간으로 인한 그것의 계수값을 여전히 1로 하여 나이를 먹는다.
4. 만일 블로크가 현재 상당히 빨리 재참조되지 않는다면 교체하는것이 좋다. 왜냐하면 새로운 구간에 있지 않는 블로크들의 참조계수값이 가장 작기때문이다. 다시 말하면 블로크들이 상대적으로 자주 참조되였다고 하더라도 그것들의 참조계수값들을 갱신하는것이 새로운 구간밖에서 나이먹고 있는 블로크들에 대하여서는 충분히 긴 시간간격으로 보이지 않는다.

이 문제를 더욱 개선하자면 다음과 같이 한다. 탄창을 새 구간, 중간구간, 낡은 구간 등의 세 구간으로 나눈다(그림 11-11 ㄴ). 앞에서와 달리 계수값은 새 구간안에 있는 블로크들에 대해서는 증가시키지 않는다. 그러나 낡은 구간안에 있는 블로크들만은 재배치를 위한 나이를 먹고 있다. 중간구간이 충분히 크다고 가정하면 이것은 상대적으로 자주 참조되는 블로크들에 치환할 나이가 되기전에 그것들의 참조값들을 갱신할 기회를 준다. 연구자들의 모의결과는 수정된 방책이 단순한 LRU나 LFU보다 훨씬 더 좋다는것을 보여 주고 있다.

특정한 치환전략에는 관계 없이 치환은 요구에 따라 또는 사전계획에 따라 발생한다. 전자의 경우에 분구는 홈이 필요될 때에만 교체된다. 후자의 경우에는 많은 홈슬로트들이 한번에 해방된다.후자의 방법의 근거는 분구들에 도로 쓰기할 필요성과 관련된다. 만일 분구를 캐쉬에로 끌어 들이고 읽기만 한다면 그것이 교체될 때 그것을 도로 디스크에 써 넣을 필요가 없다. 그러나 만일 분구가 갱신되였다면 그것을 교체하기전에 도로 써 넣을 필요가 있다. 후자의 경우에 그것은 쓰기를 밀집시켜 하도록 하며 탐색시간을 최소화하도록 쓰기한다.

성능고찰

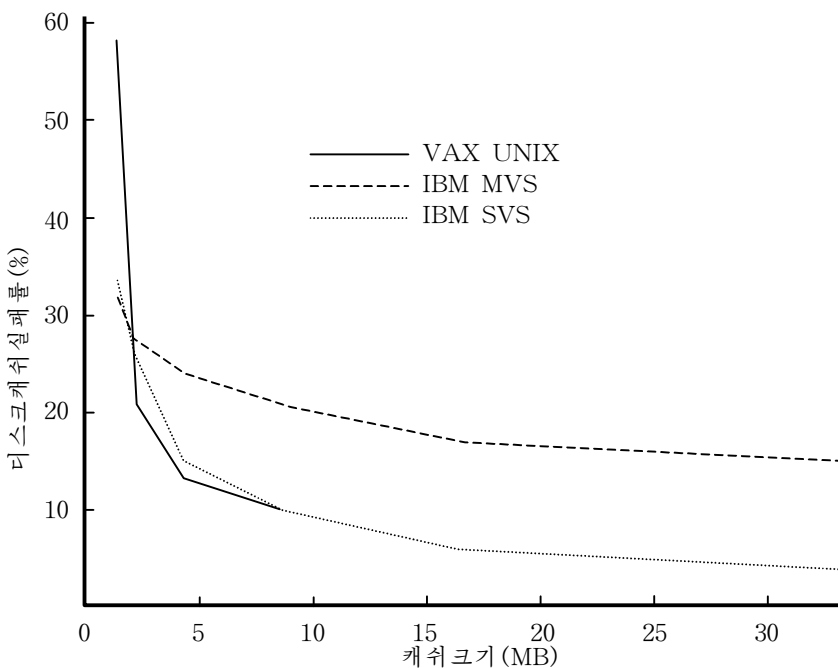



그림 11-12. LRU 법을 사용하였을 때의 디스크캐쉬성능결과

부록 1-ㄱ에서 취급한 설계고찰결과를 여기에 적용한다. 캐쉬성능문제는 주어진 실패률을 보장하는 문제로 귀착된다. 이것은 디스크참조의 국소성의 특성, 치환알고리듬 그리고 다른 설계인자들에 관계된다. 그러나 원리적으로 볼 때 실패률은 디스크캐쉬크기의 함수이다.

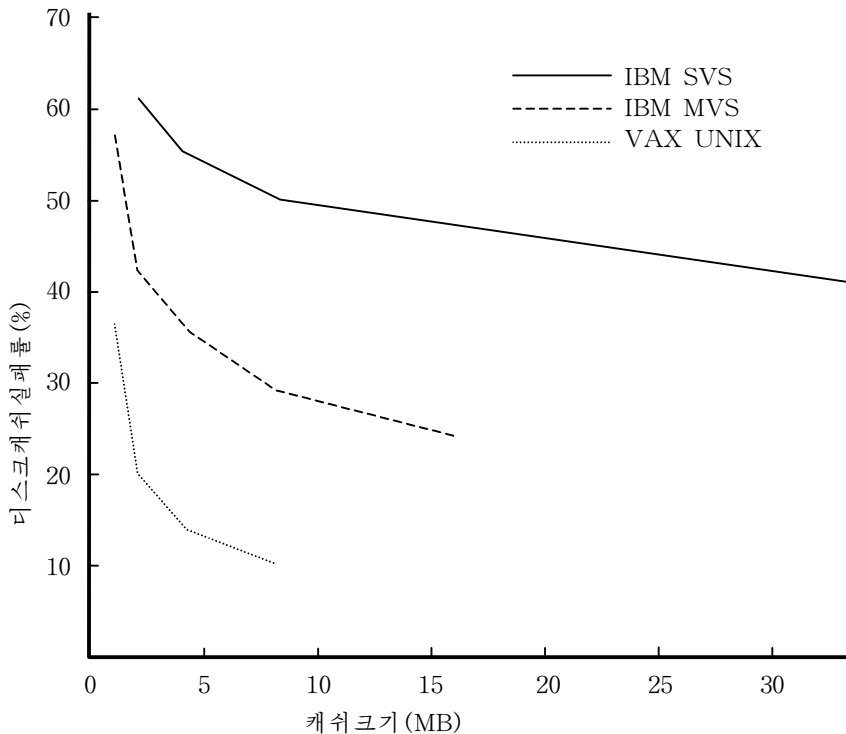



그림 11-13. 빈도에 기초한 치환방법을 사용하였을 때의 디스크캐쉬성능[ROBI90]

그림 11-12에서는 LRU를 사용한 몇가지 연구결과를 요약하였는데 거기에는 VAX에서 실행되는 UNIX체계에 대한것[OUST85]과 IBM 대형콤퓨터의 조작체계에 대한것[SMIT85]이 포함되여 있다. 그림 11-13에는 빈도에 기초한 치환알고리듬의 모의연구결과를 보여 주고 있다. 두 그림을 비교할 때 이러한 종류의 성능평가의 한가지 위험성을 보게 된다.그림에서는 LRU법이 빈도에 기초한 치환알고리듬보다 훨씬 성능이 좋은것처럼 보인다.그러나 동일한 캐쉬구조를 사용하는 꼭같은 참조패턴들을 비교하여 보면 빈도에 기초한 치환알고리듬이 더 우월하다는것을 알수 있다. 이때 참조패턴의 엄밀한 순서와 블로크크기가 같은 관련된 설계문제들은 달성된 성능에 커다란 영향을 준다.

제 8 절. UNIX SVR 4의 입출력

UNIX에서 매개 개별적인 입출력장치는 특수파일과 관련되여 있다.이것들은 파일체계에 의하여 관리되고 사용자자료관리들과 같은 방식으로 읽기 및 쓰기된다. 이것은 사용자들과 프로쎄스들에 명백하고 단일한 대면부를 준다. 장치로부터의 읽기 또는 장치에로의 쓰기를 위하여 장치와 관련되여 있는 특수파일에 읽기 및 쓰기요청을 한다.

그림 11-14에서 입출력기능의 론리적구조를 보여 주고 있다. 파일부분체계는 2차기억장치들에 있는 파일들을 관리한다. 또한 그것은 장치들에 대한 프로쎄스대면부로 봉사한다. 왜냐하면 장치는 파일과 같이 취급되기때문이다.

UNIX에는 두개의 입출력형태 즉 완충식과 비완충식이 있다. 완충식은 체계완충기를 거쳐 통과하지만 비완충식입출력은 대표적으로 입출력모듈과 프로쎄스령역사이에서 직접 이송이 진행되는 DMA기능을 요구한다. 완충식입출력에서는 두가지 형태의 완충기들 즉 체계완충기캐쉬들과 분사대기렬들을 사용한다.

완충기캐쉬

UNIX의 완충기캐쉬는 본질적으로 디스크캐쉬이다. 디스크에 의한 입출력조작들은 완충기캐쉬를 통하여 조종된다. 완충기캐쉬와 사용자프로쎄스공간사이의 자료이송은 항상 DMA를 사용하여 진행한다. 완충기캐쉬와 프로쎄스입출력명령은 다 주기억기에 있기때문에 DMA기능은 이 경우에 기억기 대 기억기복사를 진행하는데 사용된다. 이것은 임의의 처리기주기들을 다 사용하지 않지만 모선주기들은 모두 소비한다.

완충기캐쉬를 관리하기 위하여 세가지 목록이 유지된다. 즉

- **자유목록** : 배정에 사용할수 있는 캐쉬에 있는 모든 홈들의 목록(홈은 UNIX에서 완충기라고 부르며 매개 홈은 한개의 디스크분구를 담당한다.)
- **장치목록** : 현재 매개 디스크와 관련되는 모든 완충기들의 목록
- **구동기입출력대기렬** : 실제로 특정한 장치에 대한 입출력을 진행하고 있거나 기다리고 있는 완충기들의 목록

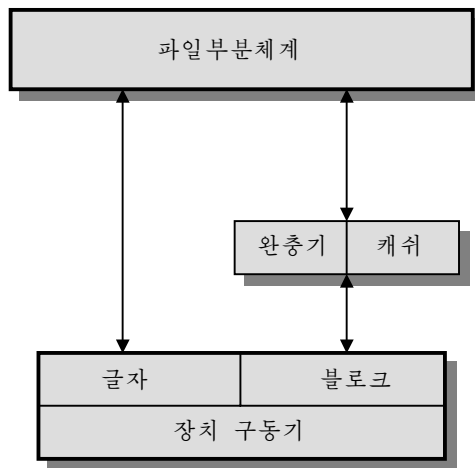



그림 11-14. UNIX 입출력구조

모든 완충기들은 자유목록 또는 장치구동기입출력대기렬에 의하여 존재한다.일단 장치와 련결된 완충기는 그것이 비록 자유목록에 있다고 해도 실제상 재사용되거나 다른 장치에 련결될 때까지 그 장치와의 련결을 계속 유지한다.이 목록은 물리적으로 떨어 져있는 목록으로서가 아니라 매개 완충기와 련결된 지시자들로서 유지된다.

특정한 장치에 대한 물리적블로크번호를 참조할 때 조작체계는 먼저 블로크가 완충기캐쉬에 있는가를 조사한다. 탐색시간을 최소로 하기 위하여 장치목록은 부록 8-ㄱ(그림 8-26 ㄴ)에서 취급한 사슬식수법을 가진 자리넘침과 류사한 수법을 사용하는 하쉬표로 구성한다. 그림 11-15에서는 완충기캐쉬의 일반적인 조직을 보여 주고 있다.

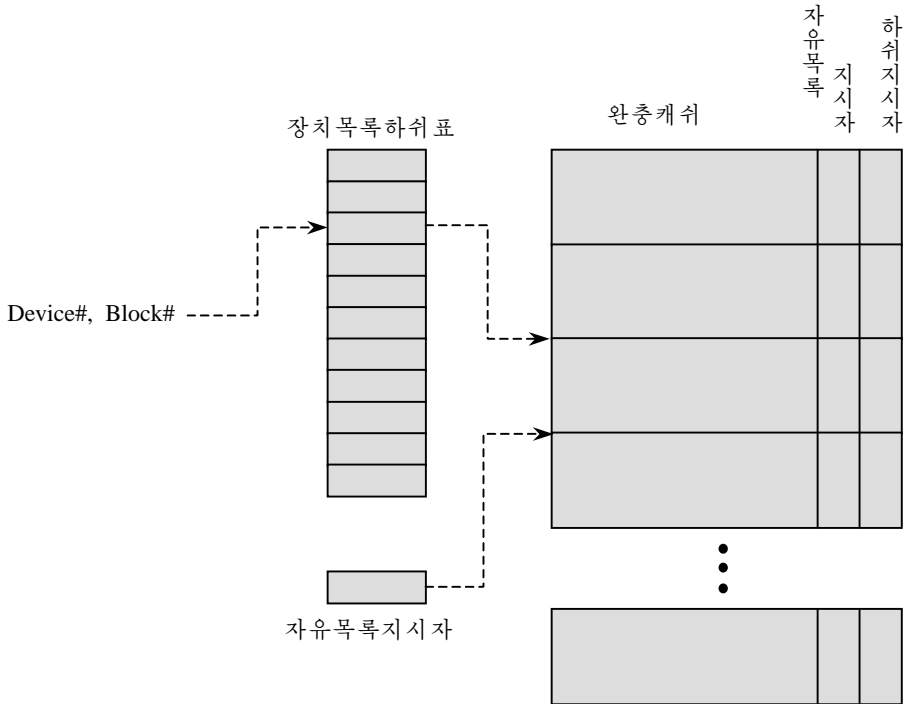



그림 11-15. UNIX 완충캐쉬조직

여기에는 완충기캐쉬를 지적하는 지시자들을 포함하는 고정길이하쉬표가 존재한다. (장치#, 블로크#)에로의 매개 참조는 하쉬표에 있는 특정한 입구점에로 사영된다. 입구점에서 지시자는 사슬에 있는 첫 완충기를 가리킨다. 매개 완충기와 관련된 하쉬지시자는 하쉬표입구점에 대한 사슬에서 다음의 완충기를 가리킨다. 따라서 같은 하쉬표입구점에로 사영하는 모든(장치#, 블로크#)참조들에서 대응하는 블로크가 완충기캐쉬에 있다면 완충기가 하쉬표입구점에 대한 사슬에 존재할것이다. 따라서 완충기캐쉬탐색길이는 거의 N분의 1배로 된다. 여기서 N은 하쉬표의 길이이다.

블로크치환에서는 최대미사용알고리듬을 사용한다. 즉 완충기가 디스크블로크에 배정된후에 그 완충기는 다른 모든 완충기들이 보다 최근에 사용될 때까지 다른 블로크용으로 사용될수 없다. 자유목록은 최대미사용순서를 보존한다.

글자대기렬

디스크나 테프와 같은 블로크지향의 장치들은 완충기캐쉬에 의하여 효과적으로 봉사를 받을수 있다. 완충화의 다른 형식은 말단이나 인쇄기들과 같은 글자지향의 장치들에 적합하다. 글자대기렬은 입출력장치에 의하여 쓰기되고 프로쎄스에 의하여 읽기되거나 프로쎄스에 의하여 쓰기되고 장치에 의하여 읽기된다. 두 경우에 모두 제5장에서 소개한 생산자/소비자모형이 사용된다. 따라서 글자대기렬들은 오직 한번만 읽어 질수 있다. 매개 글자가 읽혀 질 때 그것은 유효하게 파괴된다. 이것은 여러번 읽을수 있고 그로 인하여 읽기자/쓰기자모형(역시 제5장에서 취급한)에 따르는 완충기캐쉬와 대조적이다.

비완충식입출력

단순히 장치와 프로쎄스공간사이의 DMA인 입출력은 프로쎄스가 가장 빨리 수행할수 있게 하는 방법이다. 비완충식입출력을 수행하고 있는 프로쎄스는 주기억기에 페색되

여 있으며 교체내기될수 없다. 이것은 주기억기의 부분을 구속하는것으로 교체기회를 줄이며 따라서 전반적체계성능을 낮춘다. 또한 입출력장치는 이송기간에 프로쎄스에 구속되여 다른 프로쎄스들이 입출력장치를 사용할수 없다.

UNIX장치

UNIX는 다섯가지 형태의 장치를 인식한다.

- 디스크구동기
- 테프구동기
- 말단
- 통신선로
- 인쇄기

표 11-5는 매개 형태의 장치에 적합한 입출력형태를 보여 준다. 디스크장치는 UNIX에서 많이 사용되고 있는 블로크지향성이며 높은 처리능력을 발휘할수 있는 잠재력을 가지고 있다. 따라서 이 장치들에 대한 입출력은 비완충화하거나 완충기캐쉬를 통하여 진행하는것이 보통이다. 테프장치들은 디스크장치와 기능상 류사하며 류사한 입출력방법을 사용한다.

표 11-5. UNIX에서 장치입출력

	비완충식입출력	완충기캐쉬	글자대기렬
디스크구동기	×	×	
테프구동기	×	×	
말단			×
통신선			×
인쇄기	×		×

말단들은 상대적으로 느린 글자교환을 동반하기때문에 말단입출력에서는 대표적으로 글자대기렬을 사용한다. 이와 류사하게 통신선로들은 자료입력 또는 출력에 바이트자료의 직렬처리를 요구하며 글자대기렬들에 의하여 가장 효과적으로 조절된다. 마지막으로 인쇄기에 사용되는 입출력형태는 그것의 속도에 관계된다. 저속인쇄기들은 일반적으로 글자대기렬을 사용하며 고속인쇄기는 비완충기식입출력을 사용한다. 완충기캐쉬는 고속인쇄기에 사용할수 있다. 그러나 인쇄기로 보내 지는 자료는 결코 재사용할수 없기때문에 완충기캐쉬의 간접소비시간은 필요 없다.

제 9 절. WINDOWS 2000의 입출력

그림 11-6에서는 Windows 2000(W2K)의 입출력관리자를 보여 주고 있다. 입출력관리자는 조작체계에 대한 모든 입출력을 책임 지며 모든 형태의 구동프로그람들이 호출할수 있는 통일적인 대면부를 준다.

기본입출력모듈

입출력관리자는 4개 모듈로 구성되여 있다. 즉

- **캐쉬관리자** : 캐쉬관리자는 완전한 입출력부분체계에 대한 고속완충을 조종한다. 캐쉬관리자는 모든 파일체계와 망구성요소들에 대하여 주기억기에서 고속완충봉사를 제공한다. 그것은 사용가능한 물리기억기의 크기가 변하는데 따라 특정한 동작에 사용되는 캐쉬의 크기를 동적으로 증가 및 감소시킨다. 캐쉬관리자는 전반적인 성능을 개선하기 위한 두가지 봉사들을 모두 제공한다. 즉

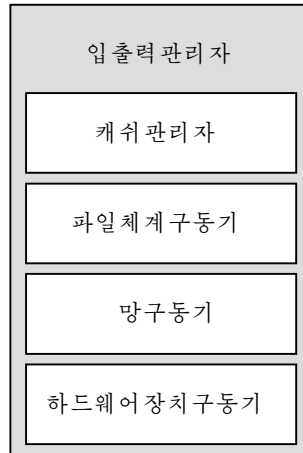



그림 11-16. Windows 2000의 입출력관리자

- **차후쓰기** : 체계는 갱신된 내용을 디스크에가 아니라 캐쉬에 기록한다. 후에 처리기에 대한 요구가 뜸해 지면 캐쉬관리자는 변화된 내용들을 디스크에 써 넣는다. 특정한 캐쉬블로크가 그동안에 갱신되면 순전히 보관만 한다.
- **차후결속** : 이것은 트랜잭션처리에 대한 차후쓰기와 류사하다. 성과적으로 완성된 트랜잭션에 즉시에 표식을 달지 않고 체계는 위탁 받은 정보를 캐쉬에 넣었다가 후에 배경프로쎄스로 파일체계기록일지에 써 넣는다.
- **파일체계구동프로그람** : 입출력관리자는 파일체계구동프로그람을 다른 장치구동프로그람으로 취급하며 그 장치적응기에 대한 적당한 쏘프트웨어구동프로그람에 일정한 기록권들에 대한 통보문을 보낸다.
- **망구동프로그람** : W2K는 통합적인 망구성능력을 가지고 있다. 분산응용들에 대한 지원을 보장한다.
- **하드웨어장치구동프로그람** : 이 구동프로그람들은 W2K집행부의 동적련결서고들에 있는 입구점들을 통하여 주변장치들의 하드웨어등록기들에 접근한다. 이 루틴들의 모임은 W2K가 지원하는 가동환경들에서 동일하기때문에 W2K장치구동프로그람들의 원천코드는 서로 다른 형태의 처리기들에 이식할수 있다.

비동기 및 동기입출력

W2K는 두가지 방식의 입출력조작 즉 비동기 및 동기입출력을 제공한다. 비동기방식은 응용의 성능을 최적화할수 있는 가능성이 있을 때 사용한다. 비동기입출력의 경우 응용은 입출력조작을 개시한 다음 입출력요청이 만족될 때까지 처리를 계속할수 있다. 동기입출력인 경우에 응용은 입출력조작이 완료될 때까지 페색된다.

비동기입출력은 호출하는 스레드의 관점에서 보면 매우 효률적이다. 왜냐하면 그것은 입출력조작이 입출력관리자에 의하여 대기렬에 넣어 지고 차례로 수행되는 동안 스레드가 집행을 계속하도록 하기때문이다. 그러나 비동기입출력조작을 기동하는 응용은 조

작이 완료되는 시간을 결정하기 위한 방도를 요구한다. W2K에서는 입출력완료를 신호하기 위한 네가지 서로 다른 수법들을 제공한다. 즉

- **장치핵심부객체신호법**: 이 방법에서는 장치객체와 관련된 지시기가 객체에 대한 조작이 완료될 때 설정된다. 입출력조작을 기동한 스레드는 입출력조작이 완료될 때까지 정지하여야 하는 점에 도달할 때까지 계속 집행할수 있다. 그 점에서 스레드는 조작이 완료될 때까지 기다렸다가 계속 집행할수 있는데 이 수법은 간단하면서도 사용자가 쉽지만 다중입출력요청들을 조종하는데는 적합하지 못하다. 실례로 스레드가 이 수법을 사용하여 파일의 어떤 부분의 읽기와 다른 부분의 쓰기와 같은 단일파일에 대한 다중동시동작을 수행하려고 한다면 스레드는 읽기의 완료와 쓰기의 완료사이의 차이를 구별할수 없다. 그저 이 파일에 대한 어떤 요청된 입출력조작이 완료되였다는것을 단순히 알수 있을뿐이다.
- **사건핵심부객체신호법** : 이 수법에서는 단일 장치 또는 파일에 대한 다중동시입출력요청들이 허락된다. 스레드는 매개 요청에 대한 사건을 생성한다. 후에 스레드는 한개의 요청 또는 완전한 요청집합에 대응한다.
- **경보가능한 입출력법** : 이 수법에서는 비동기수속호출(APC)대기렬로 알려 진 스레드와 관련된 대기렬을 사용한다. 이 경우에 스레드는 입출력요청들을 형성하며 입출력관리자는 이 요청들의 결과들을 호출중에 있는 스레드의 APC대기렬에 넣는다.
- **입출력완료포구법** : 이 수법은 스레드의 사용을 최적화하기 위하여 W2K봉사기에서 사용된다. 본질상 스레드의 집결소는 새로운 요청을 조종하는데 새로운 스레드를 창조할 필요가 제기되지 않도록 하는데 사용할 가치가 있다.

쏘프트웨어적인 RAID

W2K는 [MS96]에서 다음과 같이 정의된 두가지 종류의 RAID구성을 지원한다. 즉

- **하드웨어적인 RAID** : 디스크조종기 또는 디스크기억기하드웨어에 의하여 한개 또는 그이상의 론리디스크들로 조합된 개별적인 물리디스크들
- **쏘프트웨어적인 RAID** : 고장허용쏘프트웨어디스크구동프로그람 FTDISK에 의하여 한개 또는 그이상의 론리적분할구역들로 병합된 불련속디스크공간

하드웨어적인 RAID에서 조종기대면부는 여분정보의 창조와 재발생을 조종한다. W2K봉사기에서 사용가능한 쏘프트웨어적인 RAID기구의 쏘프트웨어는 RAID 1과 RAID 5를 실현한다. RAID 1(디스크대칭복제)의 경우에 1차 및 대칭복제된 분할구역들을 포함하는 두개의 디스크들은 동일한 디스크조종기 또는 서로 다른 디스크조종기들밑에 놓일수 있다. 후자의 구성을 디스크 2중화라고 한다.

요약, 기본용어 및 복습문제

콤퓨터체계와 외부세계와의 대면부는 곧 입출력구성방식이다. 입출력구성방식은 외부세계와의 대화를 조종하기 위한 체계적인 수단들을 제공하고 입출력동작들을 효률적으로 관리하는데 필요한 정보를 조작체계에 제공할수 있도록 설계된다.

입출력기능은 일반적으로 몇개 층으로 분할된다. 보다 낮은 층들에서는 수행하려는 물리적기능들에 보다 가까운 세부들을 취급하며 보다 높은 층들에서 론리적인 및 일반화된 방식으로 입출력을 취급한다. 결과는 하드웨어파라메터들의 변경이 대부분의 입출력쏘프트웨어에 영향을 미치지 않을것을 요구한다.

입출력의 기본문제는 응용프로쎄스들에 의해서가 아니라 입출력편의프로그람에 의하여 완충기를 조종하는 문제이다. 완충기를 사용함으로써 콤퓨터체계의 내부속도와 입출력장치들의 속도차이를 고르롭게 할수 있다. 완충기를 사용하면 실제 입출력이송을 응용프로쎄스의 주소공간으로부터 분리할수 있다. 이렇게 되면 조작체계는 자기의 기억기관리기능에서 더 많은 유연성을 보장할수 있다.

전반적인 체계성능에 가장 큰 영향을 주는 입출력문제는 바로 디스크입출력문제이다. 따라서 임의의 다른 종류의 입출력에서보다 이 령역에서 더 중요한 연구와 설계개발이 진행되여 왔다. 디스크입출력성능을 개선하기 위하여 가장 광범히 사용된 두가지 방법은 디스크일정작성법과 디스크캐쉬법이다.

임의의 시간에 같은 디스크에 대한 입출력요청들의 대기렬이 있을수 있다. 디스크의 기계적인 자리찾기시간을 최소로 하고 따라서 성능을 개선하는 방법으로 요청들을 만족시키는것이 디스크일정작성의 목적이다. 이때 현재 발생된 요청들의 물리적편성과 국소성의 고려 등의 문제가 중요하게 제기된다.

디스크캐쉬는 보통 주기억기에 위치하고 있으면서 디스크기억기와 주기억기의 잔류구역사이에서 디스크블로크들의 캐쉬로서의 기능을 수행하는 완충기이다. 국소성의 원리에 의하여 디스크캐쉬를 사용하면 주기억기와 디스크사이에서 오가는 블로크입출력의 수를 근본적으로 줄일수 있다.

기본용어

블로크	고정자두디스크	가동자두디스크
블로크지향장치	플로피디스크	비교체형 디스크
순환완충기	간격	프로그람식입출력
CD-R	하드디스크	읽기/쓰기자두
CD-ROM	새치기구동식입출력	독립디스크여분배렬
CD-RW	입출력	(RAID)
실린더	입출력완충기	분리가능디스크
수자식만능디스크(DVD)	입출력통로	회전지연시간
직접기억기접근(DMA)	입출력처리기	분구
디스크접근시간	장치입출력	자리찾기시간
디스크캐쉬	론리입출력	흐름지향장치
디스크묶음	자기디스크	자리길
	자기빛(MO)디스크	이송시간

복습문제

1. 입출력을 수행하기 위한 세가지 수법을 제시하고 간단히 정의하시오.
2. 론리입출력과 장치입출력의 차이는 무엇인가?
3. 블로크지향장치들과 흐름지향장치들사이의 차이는 무엇인가? 매개 장치에 대한 몇가지 실례를 제시하시오.
4. 입출력을 위하여 단일완충기보다 오히려 2중완충기를 사용하면 왜 개선을 기대할 수 있는가?
5. 디스크읽기 또는 쓰기에 어떤 지연요소들이 포함되는가?
6. 그림 11-8에서 설명한 디스크일정작성방책들을 간단히 정의하시오.
7. 7개의 RAID준위들을 간단히 정의하시오.
8. 대표적인 디스크분구의 크기는 얼마인가?

참 고 문 헌

[STAL00]과 [PATT98]을 비롯한 콤퓨터구성방식에 대한 많은 책들에서 콤퓨터입출력을 취급하고 있다. [MEE96a]는 디스크 및 테프체계들의 기초를 이루는 레코드작성기술에 대한 훌륭한 참고서로 된다. [MEE96b]에서는 디스크 및 테프체계들에 대한 자료기억기술을 집중적으로 취급하고 있다. [WIED87]에서는 디스크일정작성과 관련 있는 문제들과 함께 디스크성능평가문제를 구체적으로 서술하고 있다. [HG98]은 디스크하드웨어성능에 대하여 고찰하고 있다. [CAO96]에서는 디스크고속완충방법과 디스크일정작성방법을 해석하고 있다. [ROSC00]에서는 모든 형태의 외부기억체계들에 대한 기술적 세부를 적당하게 주면서 포괄적으로 개괄하고 있다. 입출력대면부에 대하여 더 많이 강조하고 장치 그자체에 대해서는 적게 취급한 책으로서는 [SCHW96]이 있다. [PAI00]은 입출력완충화와 고속완충화를 위한 통합된 조작체계기구를 교육적으로 서술한다.

[DELL00]에서는 완전한 W2K의 입출력구성방식에 대하여 잘 서술하고 있고 함께 Windows NT장치구동프로그람에 대하여 상세히 취급하고 있다.

RAID개념의 발명가들에 의하여 서술된 RAID기술에 대한 연구결과는 [CHEN94]에서 발표하고 있다. 보다 상세한 내용은 RAID관련제품의 공급자 및 소비자들의 련합인 RAID자문위원회에서 발행하고 있다. [MASS97],[CHEN96]에서는 RAID성능을 해석하고 있다. [FRIE96]도 역시 좋은 책이다. [DALT96]에서는 Windows NT쏘프트웨어 RAID기능을 상세히 설명하고 있다.

CA096 Cao, P.; Felten, E.; Karlin, A.; and Li, K. "Implementation and Performance of Integrated Application-Controlled File Caching, Prefetching, and Disk Scheduling." *A CM Transactions on Computer Systems,* November 1996.

CHEN94 Chen, P.; Lee, E.; Gibson, G.; Katz, R.; and Patterson, D. "RAID: High-Performance, Reliable Secondary Storage." *ACM Computing Surveys,* June 1994.

CHEN96 Chen, S., and Towsley, D. "A Performance Evaluation of RAID Architectures." *IEEE Transactions on Computers,* October 1996.

DALT96 Dalton, W., et al. *Windows NT Server 4: Security, Troubleshooting, and Optimization.* Indianapolis, IN: New Riders Publishing, 1996.

DELLOO Dekker, E., and Newcomer, J. *Developing Windows NT Device Drivers: A Programmer's Handbook.* Reading, MA: Addison Wesley, 2000.

FRIE96 Friedman, M. "RAID Keeps Going and Going and . . ." *IEEE Spectrum.* April 1996.

MASS97 Massiglia, P., editor. *The RAID Book: A Storage System Technology Handbook.* St. Peter, MN: The Raid Advisory Board, 1997.

MEE96a Mee, C., and Daniel, E. eds. *Magnetic Recording Technology.* New York: McGraw Hill, 1996.

MEE96b Mee, C., and Daniel, E. eds. *Magnetic Storage Handbook.* New

York: McGraw Hill, 1996.

NG98 Ng, S. "Advances in Disk Technology: Performance Issues." *Computer,* May 1989.

PAIOO Pai, V.; Druschel, P.; and Zwaenepoel, W. "10-Lite: A Unified I/O Buffering and Caching System." *ACM Transactions on Computer Systems,* February 2000.

PATT98 Patterson, D., and Hennessy, J. *Computer Organization and Design: The Hardware/Software Interface.* San Mateo, CA: Morgan Kaufmann, 1998.

ROSCOO Rosch, W. *The Winn L. Rosch Hardware Bible.* Indianapolis, IN: Sams, 2(X)0.

SCHW96 Schwaderer, W., and Wilson, A. *Understanding I/O Subsystems.* Milpitas, CA: AdaptecPress, 1996.

STALOO Stallings, W. *Computer Organization and Architecture,* 5th ed. Upper Saddle River, NJ; Prentice Hall, 2000.

WIED87 Wiederhold, G. *File Organization for Database Design.* New York: McGraw-Hill. 1987.

련 습 문 제

1. 출력장치에 접근하는 프로그람을 고찰하고 비완충입출력을 완충입출력과 비교하시오. 입출력를 사용하면 실행시간을 절반으로 줄일수 있다는것을 증명하시오.
2. 프로그람이 n개 장치를 참조하는 경우에 문제 1의 결과를 일반화하시오.
3. 다음의 디스크자리길요청순서 즉 27, 129, 110, 186, 147, 41, 10, 64, 120에 대하여 표 11-2와 같은 형태의 분석을 진행하시오. 디스크자두는 초기에 자리길 100에 위치하고 있으며 자리길번호가 감소하는 방향으로 움직이고 있다고 가정한다. 디스크자두가 자리길번호가 증가하는 방향으로 움직인다고 가정하는 경우를 분석하시오.
4. 0부터 $(N-1)$까지의 번호가 붙은 N개의 자리길을 가진 디스크를 고찰하고 요청된 분구들이 디스크에 임의로 고르롭게 분포되여 있다고 가정하자.이때 자리찾기에 의하여 횡단한 평균자리길들의 수를 계산하시오.
   ㄱ) 먼저 자두가 현재 자리길 t에 위치할 때 길이가 j인 자리찾기확률을 계산하시오. 요령: 이것은 자리찾기목적지에 대한 모든 자리길위치들이 균등하다고 보고 조합의 총수를 결정하는 문제이다.
   ㄴ) 다음으로 길이 k의 자리찾기확률을 계산하시오. 요령: 이것은 k개의 자리길들의 이동들에 대한 모든 가능한 조합들의 총합을 동반한다.
   ㄷ) 기대값에 대한 공식을 사용하여 자리찾기에 의하여 횡단한 평균자리길수를 계산하시오.

$$E[x]=\sum_{i=0}^{N-1} i\times \Pr[x=i]$$

ㄹ) N의 값이 클 때 자리찾기에 의하여 통과한 평균자리수가 $N/3$에 접근한다는 것을 증명하시오.

5. 다음의 식은 캐쉬기억기와 디스크캐쉬기억기에 대해서 모두 성립한다. 즉

$$Ts=Tc+M \times T_D$$

이 식을 2준위대신 N준위의 계층기억기에로 일반화하시오.

6. 빈표에 기초한 알고리듬(그림 11-12)에서 새로운, 중간의, 낡은 구역들로 구성된 캐쉬의 부분들로서 F_{new}, F_{middle}, F_{old}를 각각 정의하시오. 명백히F_{new} + F_{middle} + F_{old} = 1이다.

ㄱ) F_{old} = 1-F_{new}
ㄴ) F_{old} = 1/캐쉬크기
일 때 방책의 특징을 설명하시오.

7. 테프속도가 초당 120인치이고 테프밀도는 인치당 1600선형비트인 9자리길자기테프장치의 전송속도는 얼마인가?

8. 테프가 읽기사이의 중간에서 정지하는 레코드사이간격이 0.6인치인 2400피트테프감개가 있다. 테프속도는 간격이 선형적으로 증가/감소하며 테프의 다른 특성지표들은 문제 5에서와 같다고 가정하자. 테프에 기록된 자료는 매개가 론리레코드라고 하는 고정된 수의 사용자정의단위들을 포함하는 물리적레코드들로 구성되여 있다.

ㄱ) 물리레코드당 120byte론리블로크 10개씩 블로크화된 테프를 완전히 읽는데 얼마의 시간이 걸리는가?
ㄴ) 30개씩 블로크화된 경우에는 얼마의 시간이 걸리는가?
ㄷ) 우에서 지적한 매개 블로크작성요소들로는 얼마나 많은 론리레코드들을 유지할수 있는가?
ㄹ) 우에서 지적한 매개 블로크작성요소들에 대한 총체적인 실효이송속도는 얼마인가?
ㅁ) 테프의 용량은 얼마인가?

9. 디스크가 512byte/분구인 고정분구형식이면 련습문제 6, 7에서 읽은 론리레코드들을 보관하는데 얼마나 많은 디스크공간(분구, 자리길, 면)이 요구되는가를 계산하시오.

10. 련습문제 7에서 설명한 디스크체계를 고찰하자. 디스크는 360회/min속도로 회전한다. 처리기는 바이트당 한개의 새치기를 가지는 새치기구동식입출력을 사용하여 디스크에서 한개의 분구를 읽는다. 매개 새치기를 처리하는데 2.5s가 걸린다면 처리기가 입출력을 처리하는데 소비하는 시간몫은 얼마인가(자리찾기시간은 고려안함)?

11. 분구당 한개의 새치기를 가진다고 가정하고 DMA를 사용하여 련습문제 8을 다시 푸시오.

12. 32bit콤퓨터가 두개의 선택통로와 한개의 다중선택통로를 가지고 있다. 매개 선택통로는 두개의 자기디스크와 두개의 자기테프장치를 지원한다. 다중선택통로는 두개의 행인쇄기, 두개의 카드읽기장치와 거기에 련결된 10개의 VDT말단을 가지고 있다. 이송속도는 다음과 같다고 하자. 즉

디스크구동	800kbytes/s
자기테프구동	200kbytes/s
행인쇄기	6.6kbytes/s
카드읽기장치	1.2kbytes/s
VDT	1kbytes/s

이 체계에서 총적인 최대입출력이송속도를 계산하시오.

13. 기억띠의 크기가 입출력요청의 크기보다 작을 때 디스크의 기억띠화가 자료이송 속도를 개선할수 있다는것은 명백하다. 또한 RAID 0은 단일한 대형디스크에 비해 상대적으로 성능이 개선되였다는것도 명백하다. 그것은 여러개의 입출력요청을 병렬로 처리할수 있기때문이다. 그러나 후자의 경우에 디스크의 기억띠화가 필요한가? 다시 말하여 디스크의 기억띠화가 그것이 없는 비교할만한 디스크배렬에 비해 입출력요청의 속도성능을 개선하여 주는가?

부록 11-ㄱ. 디스크기억장치

자기디스크

디스크는 자성재료를 입힌 금속이나 수지로 된 원형회전판이다. **자두**라고 부르는 전도코일을 통하여 자료를 디스크에 기록하거나 후에 디스크에서 다시 회복한다. 읽기 또는 쓰기조작기간에 자두는 움직이지 않고 회전판이 그 밑에서 돌아 간다.

쓰기방법은 코일을 통하여 흐르는 전기가 자기마당을 발생시키는 원리에 기초하고 있다. 임풀스들을 자두에 보내면 자기적인 패턴들이 자두밑의 표면에 기록되는데 정 및 부의 전류에 의하여 서로다른 패턴들이 형성된다. 읽기방법은 코일에 대하여 상대적으로 움직이는 자기마당이 그 코일에 전류를 발생시킨다는 사실에 기초하고 있다. 디스크의 표면이 자두밑을 통과할 때 그것은 이미 기록된것과 같은 극성의 전류를 발생시킨다.

자료조직과 양식화

자두는 그 밑에서 돌아 가는 기록판의 일부분에서 자료를 읽거나 거기에 써 넣을수 있는 상대적으로 작은 장치이다. 이것은 자료가 **자리길**이라고 부르는 밀집된 동심고리들의 모임으로 원판우에 구성되게 한다. 매개 자리길의 너비는 자두의 너비와 같다. 한면에는 수천개의 자리길이 있다.

그림 11-17에서는 이 자료의 배치상태를 보여 주고 있다. 린접한 자리길들은 **간격**을 두고 분리되여 있다. 이것은 자두를 잘못 설치하거나 단순히 자기마당의 간섭으로 인하여 생기는 오유를 막아 주거나 아니면 적어도 최소로 되게 한다. 전자회로적처리를 간단히 하기 위해 매개 자리길에는 표준적으로 같은 수의 비트들을 기억시킨다. 그러므로 인치당 비트수로 표시하는 **밀도**는 바깥 자리길로부터 안쪽 자리길로 움직임에 따라 증가한다(이와 같은 현상은 구식축음기기록에서 볼수 있다.).

자료는 디스크안팎으로 **블로크들**로 이송된다. 보통 블로크는 자리길의 용량보다 작다. 따라서 자료는 **분구**라고 하는 블로크의 크기만한 구역들에 기억된다(그림 11-17).대체로 자리길마다 몇백개의 분구가 있는데 이것은 고정길이로 될수도 있고 가변길이로 될수도 있다. 대부분의 디스크구동기들에서 고정분구의 크기로서는 512byte를 사용하고 있다. 체계에 과도한 정밀도요구를 지우지 않기 위해 린접분구들은 자리길사이(분구사이)간격들로 분리된다.

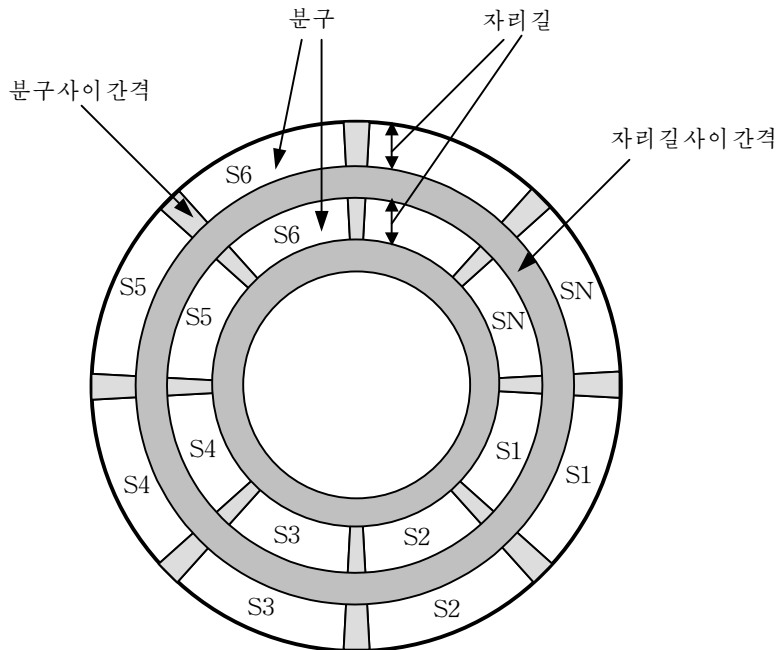



그림 11-17. 디스크자료의 배치

자리길안에서 분구의 위치를 찾는데 필요한 어떤 수단들이 있어야 한다. 분명 자리길의 시작점과 매개 분구의 시작과 끝을 식별하는 방법이 있어야 한다. 이 요구들은 디스크에 기록된 조종자료에 의하여 처리된다. 그러므로 디스크는 디스크구동에만 사용되고 사용자는 접근할수 없는 일정한 여분의 자료로 양식화된다.

물리적특성

표 11-6에서는 여러가지 형태의 자기디스크들을 구별할수 있는 주되는 특별지표들을 렬거하고 있다. 우선 자두를 기록판의 반경방향에 대하여 고정시킬수도 있고 이동시킬수도 있다. **고정자두디스크**에는 자리길마다 한개의 읽기/쓰기자두가 있다. 그 모든 자두들은 모든 자리길들을 가로 질러 고정된 팔에 설치된다. **가동자두디스크**에는 읽기/쓰기자두가 한개만 있다. 그 자두도 어떤 팔에 설치된다. 자두를 임의의 자리길우에 배치할수 있어야 하므로 이를 위해 팔을 늘였다줄였다 할수 있다.

자리길안에서 분구의 위치를 찾는데 필요한 어떤 수단들이 있어야 한다. 분명 자리길의 시작점과 매개 분구의 시작과 끝을 식별하는 방법이 있어야 한다. 이 요구들은 디스크에 기록된 조종자료에 의하여 처리된다. 그러므로 디스크는 디스크구동에만 사용되고 사용자는 접근할수 없는 일정한 여분의 자료로 양식화된다.

물리적특성

표 11-6에서는 여러가지 형태의 자기디스크들을 구별할수 있는 주되는 특별지표들을 렬거하고 있다. 우선 자두를 기록판의 반경방향에 대하여 고정시킬수도 있고 이동시킬수도 있다. **고정자두디스크**에는 자리길마다 한개의 읽기/쓰기자두가 있다. 그 모든 자두들은 모든 자리길들을 가로 질러 고정된 팔에 설치된다. **가동자두디스크**에는 읽기/쓰기자

두가 한개만 있다. 그 자두도 어떤 팔에 설치된다. 자두를 임의의 자리길우에 배치할수 있어야 하므로 이를 위해 팔을 늘였다줄였다 할수 있다.

디스크자체는 디스크구동기에 설치되는데 그것은 팔, 디스크를 회전시키는 주축 및 2진자료를 입출구하기 위하여 필요한 전자회로로 구성되여 있다. **비교체형디스크**는 디스크구동기에 영구적으로 설치된다. **분리가능디스크**는 분리하여 다른 디스크와 교체할수 있다. 분리가능디스크의 우점은 제한된 수의 디스크체계로 제한없는 자료를 사용할수 있다는것이다. 또한 그러한 디스크를 하나의 콤퓨터체계에서 다른 콤퓨터체계에로 이동시킬수 있다.

대부분의 디스크들에서 자성도포는 기록판의 량쪽면에 자성체를 도포하는데 그것을 **량면디스크**라고 한다. 일부 값눅은 디스크체계들에서는 **단면디스크**를 사용한다.

일부 디스크구동기들은 어떤 간격으로 수직으로 쌓은 **다중회전판들**을 관리한다. 여기에는 여러개의 팔이 설치되여 있다. 회전판들은 **디스크묶음**이라고 하는 하나의 장치로 된다(그림 11-18). 다중회전판디스크는 이동식자두를 사용하는데 회전판의 면마다 한개의 읽기/쓰기자두를 가지고 있다. 모든 자두는 기계적으로 고정되여 있어 모두가 디스크의 중심으로부터 같은 거리에 위치하고 있고 동시에 움직인다. 따라서 임의의 시간에 모든 자두들은 디스크의 중심에서 같은 거리에 있는 자리길우에 놓이게 된다. 회전판우에서 같은 상대적 위치에 있는 모든 자리길들의 모임을 **실린더**라고 한다. 실례로 그림 11-19에서 어둡게 표시한 모든 자리길들이 한개의 실린더의 일부분으로 된다.

끝으로 자두설치방법은 세개의 형태로 분류한다. 전통적으로 읽기/쓰기자두는 회전판우에서 고정된 거리에 위치하고 있으면서 일정한 높이의 공기간격을 보장하고 있다. 또한 읽기 또는 쓰기동작시간에 매질과 실제적으로 물리적접촉을 하도록 자두를 설치하는 방법이 있다. 이 방법을 **플로피디스크**에 사용하는데 이것은 크기가 작고 유연한 회전판으로서 원가가 최소인 디스크로 된다.

세번째 형태의 디스크를 리해하기 위해서는 자료밀도와 빈 간격의 크기사이관계에 대하여 설명해야 한다. 자두의 쓰기 및 읽기를 알맞게 하는데 충분한 크기의 전자기마당을 발생하거나 수감하여야 한다. 자두가 좁을수록 회전판의 면에 더 가까이 접근하여 동작해야 한다. 자두가 좁다는것은 자리길이 좁다는것을 의미하며 따라서 자료밀도가 더 커진다는것을 의미한다. 그러나 자두가 디스크에 더 가까이 접근할수록 불순물이나 미완성가공으로 인한 오유의 위험은 더 커진다. 기술적으로 더 안받침하기 위해 윈체스터디스크를 개발하였다. 윈체스터자두는 거의나 오물이 없는 봉인된 구동기조립품으로 사용된다.

표 11-6. 디스크체계의 물리적특성지표

자두운동	**기록판**
고정자두(자리길마다 한개)	단일회전판
가동자두(면마다 한개)	다중회전판
디스크휴대성	**자두설치방법**
비교체형디스크	접촉식(유연성)
분리가능디스크	고정간격식
면	기체력학적간격식(윈체스터)
단면	

그것들은 디스크의 표면에 종전의 고정식디스크자두보다 더 가까이에서 조작하도록 설계되여 있으므로 더 큰 자료밀도를 보장하고 있다. 그 자두는 사실상 디스크가 운동하지 않을 때 회전판의 면우에 가볍게 놓이는 기체력학식 엷은 판이다. 회전하는 디스크에 의해 발생되는 공기압력은 그 엷은 판을 표면우로 들어 올리는데 충분하다. 이러한 접촉체계에서는 종전의 고정식디스크자두보다 회전판의 면에 더 가까이 접근하여 동작하는 보다 좁은 자두를 사용할수 있다.[4]

표 11-7에는 대표적인 고성능이동자두식디스크의 디스크구동기의 파라메터들을 제시하였다.

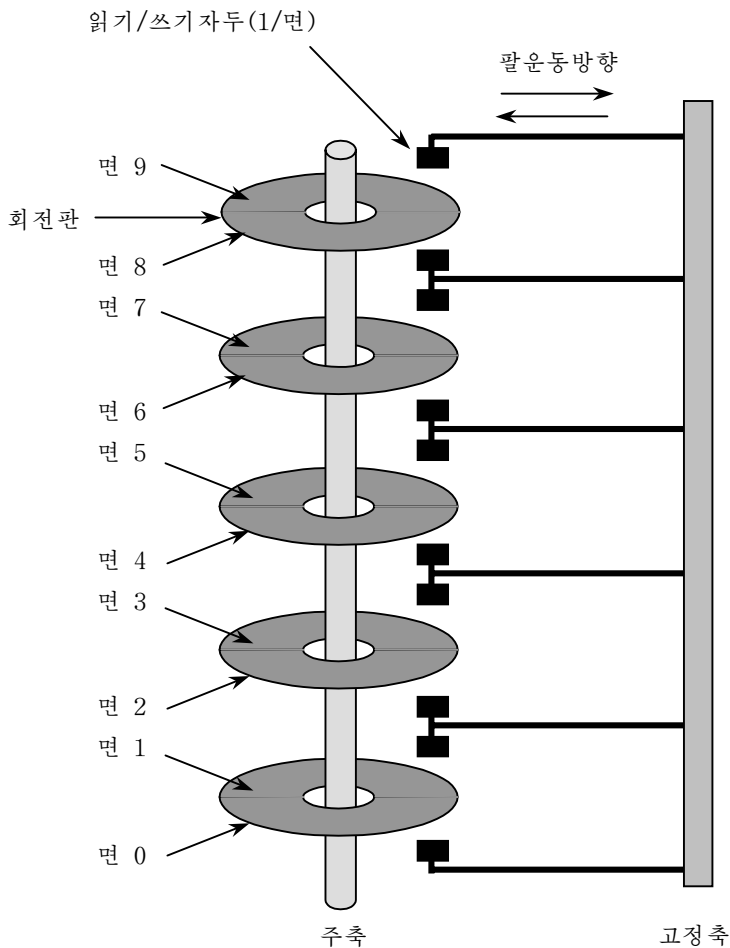



그림 11-18. 디스크구동기의 요소

빛기억기

[4] 개발과정을 돌이켜 볼 때 사실 윈체스터라는 용어는 원래 IBM회사가 발표하기전에 3340디스크모형에 대한 코드이름으로 사용하였다. 3340은 묶음내에 밀폐된 자두를 가진 교체할수 있는 디스크묶음이였다. 이제부터 이 용어가 기체력학식 자두로 설계된 임의의 밀폐장치형 디스크구동기들에 적용되고 있다. 윈체스터디스크는 일반적으로 개인용콤퓨터나 작업기들에 만들어 넣은것을 보게 되는데 여기서는 하드디스크라고 부른다.

1983년에 그때까지의 가장 성공적인 소비품중의 하나인 콤팍트디스크(CD)수자음성체계가 소개되였다. CD는 한면에 60분이상의 음성정보를 기억할수 있는 지울수 없는 디스크이다. CD의 거대한 상업적성공은 낮은 비용의 빛디스크기억방법을 개발할수 있게 하였으며 그것은 콤퓨터의 자료기억에서 혁명이였다. 여러가지의 빛디스크체계를 표 11-8에 제시하였다. 여기서는 매개 빛디스크에 대하여 설명한다.

CD-ROM

음성 CD와 CD-ROM(읽기전용고밀도디스크)은 모두 류사한 수법을 적용하고 있다. 기본차이는 CD-ROM읽기장치들이 보다 정교하지 못하여 오유정정장치를 가지고 디스크로부터 콤퓨터에로 적절히 이송되도록 하는것이다. 두가지 형태의 디스크는 또한 제작방법도 동일하다. 디스크는 폴리카보네이트와 같은 수지로 성형하고 보통 알루미니움으로 된 고도의 반사면을 도포한다. 수자식으로 기록된 정보(음악이든 콤퓨터자료이든)는 반사면우에 미세한 구멍계렬로 찍힌다. 우선 미세하게 집초된 세기가 강한 레이자를 사용하여 주디스크를 만든다. 그 우에 투명락카를 도포하여 홈들이 찍힌 복사면을 먼지나 긁힘으로부터 보호한다.

빛디스크읽기장치에 내장되여 있는 낮은 출력의 레이자 즉 구동장치가 CD 또는 CD-ROM에서 정보를 꺼낸다. 레이자는 투명한 보호도포막을 투과하여 비치며 전동기는 디스크가 그것을 통과하도록 회전시킨다. 반사된 레이자빛의 세기는 홈에 맞다 들릴 때마다 변한다. 이 변화를 빛수감기로 검출하여 수자신호로 변환한다.

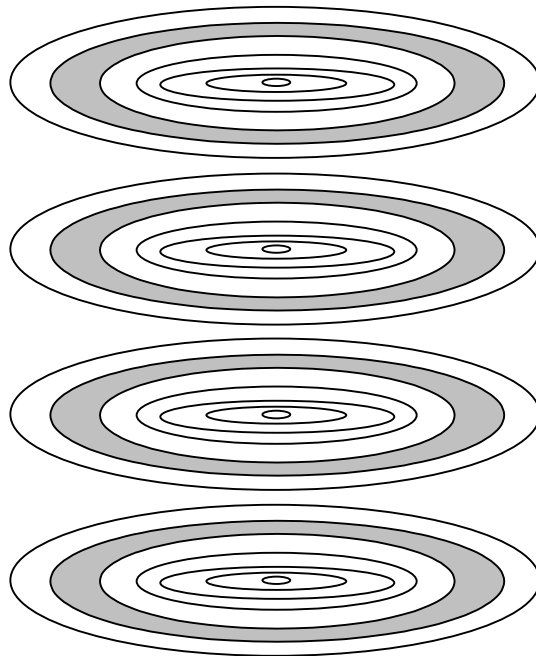



그림 11-19. 자리길과 실린더

표 11-7. 대표적인 디스크구동기의 파라메터

특성지표	Cheetah36	WDE18300
용량	36.4GB	18.3GB
최소자리길 대 자리길 찾기시간	0.6ms	0.6ms
평균자리길찾기시간	6ms	5.2ms
주축속도	10000회/min	10000회/min
평균회전지연	3ms	3ms
최대이송속도	313Mbps	360Mbps
분구당 바이트수	512	512
자리길당 분구수	300	320
실린더당 자리길수 (회전판의 면수)	24	8
실린더수(회전판의 한면우에 있는 자리길의 수)	9801	13614

회전하는 디스크중심부근의 홈은 바깥쪽에 있는 홈보다 더 천천히 고정된 점(레이자 광속과 같은)을 지나가게 하고 이때 레이자가 모든 홈들을 같은 속도로 읽을수 있도록 하기 위하여 속도변화를 보상하는 방법을 찾아야 한다. 이것은 자기디스크들에서와 마찬가지로 디스크의 토막들에 기록되는 정보의 비트들사이에 공백들을 증가시켜 해결할수 있다. 다음 **등각속도(CAV)**라고 하는 고정된 속도로 디스크를 회전시켜 동일한 속도로 정보를 주사할수 있다. 그림 11-20 ㄱ에서는 CAV를 사용하는 디스크의 배치를 보여주고 있다. 디스크를 많은 분구들과 동심자리길들의 계렬로 나눈다. CAV를 사용하는 우점은 자리길과 분구에 의하여 개별적인 자료블로크들을 직접 주소지정할수 있는것이다. 자두를 현재의 위치에서 특정한 주소위치에로 이동시키자면 다만 자두를 특정한 자리길로 짧게 이동시키고 해당한 분구가 회전하여 자두밑에 올 때까지 잠간 기다리면 된다. CAV의 결함은 긴 바깥자리길에 기억시킬수 있는 자료의 량이 짧은 내부자리길에 기억시킬수 있는 자료의 량과 같다는것이다.

디스크의 바깥쪽에 보다 적은 정보를 배치하는것은 공간을 랑비하기때문에 CD와 CD-ROM에서는 CAV법을 사용하지 않는다. 그대신 정보를 동일한 크기의 토막들을 디스크에 고르롭게 채우고 디스크를 가변적인 속도로 회전시키면서 동일한 속도로 이것을 주사한다. 다음 **등선속도(CLV)**로 레이자에 의해 홈들을 읽는다. 디스크는 중심근방보다 바깥쪽 근방에 접근할 때 더 천천히 회전한다. 그러므로 디스크의 바깥쪽 근방에있는 자리길위치에 대하여 자리길의 용량과 회전지연이 모두 증가한다.

각이한 밀도를 가진 CD-ROM들이 생산되였다. 대표적실례로 자리길과 자리길사이 간격은 1.6μm(1.6$\times 10^{-6}$m)이다. CD-ROM에서 반경에 따르는 기록폭은 32,550μm이고 결국 총체적인 자리길수는 32.550μm를 자리길의 간격으로 나눈것 즉 20344개로 된다. 사실상 단일한 라선형자리길이 있고 평균원주에 라선의 감기수를 곱하여 자리길의 길이를 계산해 낼수 있는데 이것은 대략 5.27km에 달한다. CD-ROM의 등선속도는 1.2m/s인데 이것은 총 4391s 즉 73.2min에 해당하며 이것은 대략 음성용 밀집형디스크의 표준최대동작시간으로 된다. 자료가 디스크로부터 176.4kbyte/s로 흘러 나오므로 CD-ROM의 기억용량은 774.57Mbyte이다. 이것은 550개의 3.25인치 디스케트와 맞먹는다.

그림 11-20 ㄴ에서는 CD와 CD-ROM에서 사용한 등선속도방식으로 배치한 상태를 보여 주고 있다. CLV를 사용하면 우연접근이 좀 힘들어 진다. 특정한 주소를 찾자면 자두를 일반적인 구역으로 이동시키고 회전속도를 조절하여 그 주소를 읽어야 하며 다음 특정한 분구를 찾고 그에 접근하기 위한 미세조절을 해야 한다.

CD-ROM은 대량적인 자료를 많은 사용자들에게 보급하는 경우에 합리적이다. 초기 써넣기과정이 비싸기때문에 개별적으로 응용하는데서는 적합하지 못하다. 전통적인 자기디스크에 비하여 CD-ROM은 다음과 같은 세가기 중요한 우점을 가지고 있다. 즉

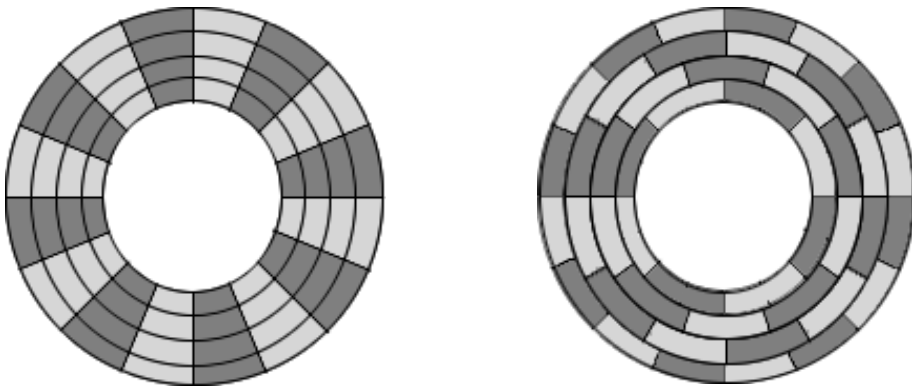



그림 11-20. 디스크배치법의 비교
ㄱ-등각속도, ㄴ-등선속도

- 정보기억용량이 빛디스크보다 훨씬 크다.
- 빛디스크는 그것에 기억된 정보를 자기디스크와는 달리 값눅게 대량적으로 재현할수 있다. 이때 자기디스크의 자료기지는 두개의 디스크구동기를 사용하면서 한번에 하나의 디스크를 복사하여 재현해야 한다.
- 빛디스크는 교체할수 있으며 디스크자체를 기록보관용 기억기로 쓸수 있다. 대부분의 자기디스크는 교체할수 없다. 비교체형 자기디스크의 정보는 우선 테프에 복사해야 하며 그다음 디스크구동기/디스크를 새로운 정보를 기억시키는데 사용할수 있다.

CD-ROM의 결함은 다음과 같다. 즉

- 읽기전용이며 갱신할수 없다.
- 접근시간이 자기디스크구동기의 접근시간보다 훨씬 길어서 0.5s 정도 걸린다.

기록가능 CD

어떤 자료의 모임에 대한 단 한번 또는 적은 회수의 복사가 필요한 응용분야들에서의 편의를 도모하기 위해 기록할수 있는 CD(CD-R)라고 하는 한번 쓰고 여러번 읽을수 있는 CD가 개발되였다. CD-R는 적당한 세기를 가지는 레이자광속으로 후에 한번 써넣을수 있게 준비되여 있는 디스크이다. 그러므로 CD-ROM보다 어느정도 비싼 디스크조종기를 사용하여 사용자는 한번 써 넣을수 있고 또 여러번 디스크를 읽을수 있다.

CD-R매질은CD나 CD-ROM의 매질과 류사하지만 꼭 같지는 않다. CD와 CD-ROM에서는 매질의 표면에 홈을 내여 반사능력을 변화시켜 정보를 기록한다. CD-R에서는 매질이 착색층을 가지고 있다. 반사능력을 변화시키는데 착색제를 사용하며 높은 세기를 가진 레이자로 활성화시킨다. 그렇게 만들어 지는 디스크를 CD-R구동기나 CD-ROM구동기에서 읽을수 있다.

CD-R빛디스크는 서류와 파일들을 기록보관소에 기억시키는데서 아주 쓸모가 있다. 그것은 대용량의 사용자자료를 영구적으로 기록시킨다.

표 11-8. 빛디스크제품

CD
밀집형디스크. 수자화된 음성정보를 기억하는 지울수 없는 디스크. 표준체계는 12cm 디스크를 사용하여 60분이상의 중단없는 연주시간을 기록할수 있다.
CD-ROM
읽기전용밀집형디스크. 콤퓨터자료를 기억하는데 사용하는 지울수 없는 디스크. 표준체계는 12cm 디스크를 사용하여 650Mbyte이상 유지할수 있다.
CD-R
쓰기가능 CD. CD-ROM과 류사하다. 사용자가 디스크에 한번 써넣을수 있다.
CD-RW
재쓰기가능 CD. CD-ROM과 류사하다. 사용자가 1000번까지 디스크를 지우고 다시 쓸수 있다.
DVD
수자식만능디스크. 영상정보의 수자화, 압축표현은 물론 대용량의 수자자료를 발생시킬수 있다. 8, 12cm 의 직경을 가진것을 사용하며 량면은 15.9Gbyte까지의 용량을 가진다. 기본적인 DVD는 읽기전용 (DVD-ROM)이다.
DVD-R
재쓰기가능 DVD. DVD-ROM과 류사하다. 사용자디스크에 한번만 써넣을수 있다.
DVD-RW
재쓰기가능한 DVD. DVD-ROM과 류사하다. 사용자가 1000번까지 디스크에 지우고 재쓰기할수 있다.
자기-빛디스크
읽기에서 빛기술과 빛을 집초하는 자기기록수법을 사용하는 디스크. 3.5인치 및 5.25인치 디스크를 모두 사용한다. 용량은 5Gbyte이상이 보통이다.

재쓰기가능 CD

CD-RW빛디스크는 자기디스크와 같이 여러번 쓰고 덧 쓸수 있다. 많은 방법을 시도하였으나 순수한 빛방법을 (후에 설명하는 자기빛방법과 달리) 상변화법이라고 부른다. 상변화식디스크는 두개의 서로다른 상의 상태에서 두가지의 크게 차이나는 반사능력을 가진다. 분자들이 무질서하게 배치되여 빛을 약하게 반사시키는 무정형상태가 있으며 빛을 잘 반사시키는 평탄한 면을 가지는 결정상태가 있다. 레이자광속의 재료를 한 상으로부터 다른 상에로 변화시킬수 있다. 상변화식빛디스크의 기본결함은 재료가 종당에 가서는 자기의 희망하는 특성을 상실하게 된다는것이다. 현재의 재료는 500,000 ~1,000,000번의 지우기주기를 거듭하면서 사용할수 있다.

CD-RW는 재쓰기할수 있고 그런데로부터 실제적인 2차기억기로 사용할수 있기때문에 CD-ROM과 CD-R에 비하여 뚜렷한 우점을 가지고 있다. 그렇기때문에 CD-RW는 자기디스크와 경쟁하고 있다. 지우기가능 빛디스크의 자기디스크에 비한 기본적인 우점은 다음과 같다. 즉

- **고용량**: 5.25인치의 빛디스크는 대략 650Mbyte의 자료를 보관할수 있다. 가장 발전된 윈체스터디스크는 이 량의 절반보다 더 적게 보관할수 있다.
- **휴대성**: 빛디스크를 구동기에서 교체할수 있다.

- **믿음성**: 빛디스크에 대한 공학적인 허용오차는 용량이 큰 자기디스크에서보다 훨씬 덜 심각하다. 그러므로 그것들은 믿음성이 높고 수명이 길다.

수자식만능디스크

전자공업의 발전으로 대용량 수자식만능디스크(DVD)를 사용하여 상사식 VHS비데오테프를 교체할수 있게 되였다. 앞으로 DVD는 비데오카세트록화기(VCR)에서 사용하는 비데오테프를 교체하게 될것이며 보다 중요하게는 개인용 콤퓨터와 봉사기들에서 쓰이는 CD-ROM을 교체하게 될것이다. DVD는 비데오를 수자식시대에로 이끌어 가고 있다. 그것은 높은 화질을 가진 영화들을 배포하고 있으며 음성 CD와 같이 자유롭게 호출할수 있으며 DVD기계도 역시 동작시킬수 있다. 많은 권수의 자료를 디스크에 채워 넣을수 있는데 현재 CD-ROM의 7배에 달하고 있다. 앞으로 DVD의 대용량과 선명한 질을 사용하여 개인용 콤퓨터에 의한 오락들이 보다 현실감을 주게 될것이며 교육쏘프트웨어가 비데오를 더 많이 포함하게 될것이다. 이 자료들을 Web싸이트들에 삽입하면 인터네트와 협동인트라네트들에서는 이 발전에 뒤를 이어 새로운 정보흐름의 비약이 일어 날것이다.

여기에 DVD를 CD-ROM과 구별해 주는 몇가지 기본특징점들이 있다. 즉

- 표준 DVD는 층당 4.7Gbyte를 보관하며 2층단일면의 DVD는 8.5Gbyte를 보관한다.
- DVD는 질높은 화면을 얻기 위해 MPEG라고 하는 비데오압축형식을 사용한다.
- 단일층 DVD는 2시간 13분의 영화를 보관할수 있으며 2층비데오는 4시간이상의 긴 영화를 보관할수 있다.

자기빛디스크

자기빛(MO)디스크구동기는 전통적인 자기디스크체계의 용량을 증가시키기 위하여 빛레이자를 사용한다. 기록수법은 기본상 자기적수법이다. 그러나 빛레이자를 사용하는데 사실은 자기식기록자두에 주의를 돌려 더 큰 용량을 얻을수 있게 하자는것이다. 이 방안에서는 디스크를 높은 온도에서만 그것의 극성을 변경시킬수 있는 재료로 도포한다. 레이자를 사용하여 표면우의 미세한 점을 가열시키고 다음에 자기마당을 가해 주는 방법으로 디스크우에 정보를 써 넣는다. 그 점이 식을 때 그것은 마당의 북남극성을 가지게 된다. 극성화처리는 디스크에서 물리적변화를 일으키지 않으므로 처리를 여러번 반복할수 있다.

읽기조작은 순수 빛으로만 한다. 자화방향은 극성을 가진 레이자빛(쓰기조작때보다 낮은 출력을 가진다.)으로 검출할수 있다. 특정한 점에서 반사되는 극성을 가진 빛은 자기마당의 방향에 따라 회전각도를 변화시킨다.

순수한 빛 CD구동기에 비한 MO구동기의 기본우점은 디스크의 수명이다. 빛디스크에서는 자료를 반복하여 쓰면 점차 매질의 감퇴를 일으킨다. 그러나 MO구동기에서는 이러한 감퇴현상이 일어 나지 않으며 따라서 반복재쓰기를 하면서 계속 사용할수 있다. MO구동기의 또 하나의 우점은 자기기억기에 비하여 메가바이트당 비용이 매우 적은것이다.

제 12 장. 파일관리

대부분의 응용들에서 파일은 중요한 요소이다. 실시간응용들과 몇가지 다른 전용화된 응용들을 제외하고 응용프로그람의 입력은 파일로 진행되며 실제상 모든 응용프로그람들에서 출력은 장기적인 기억을 위하여 그리고 사용자와 다른 프로그람들에 의한 앞으로의 접근을 위하여 파일에 보관된다.

파일은 입력과 출력을 위하여 파일을 사용하는 임의의 개별적인 응용을 제외하고 수명을 가진다. 사용자들은 파일에 접근하고 그것을 보관하며 그 내용을 보존하려고 한다. 이 목적을 위하여 실제 모든 조작체계들은 파일관리체계들을 제공한다. 대표적으로 파일관리체계는 특권이 있는 응용들로서 실행하는 체계유틸리티들로 구성된다. 그러나 파일관리체계는 조작체계로부터 특수봉사들을 요구한다. 완전한 파일관리체계는 기껏해서 조작체계의 한부분으로 간주된다. 따라서 이 책에서 파일관리의 기본요소들을 고찰하는것이 합리적이다.

이 장에서는 먼저 파일관리에 대하여 개괄하고 여러가지 파일조직방법들에 대하여 고찰한다. 파일조직은 일반적으로 조작체계의 범위를 벗어 나지만 파일관리에 필요한 여러가지 설계교환조건들을 평가하기 위하여 공통적인 방법들에 대한 일반적인 리해를 가지는것이 중요하다. 이 장의 나머지 부분에서는 파일관리의 다른 원리들을 고찰한다.

제 1 절. 개괄

파일

파일을 론의할 때 4개의 용어를 공통적으로 사용한다.

- 마당
- 레코드
- 파일
- 자료기지

마당은 자료의 기본요소이다. 개별적인 마당은 종업원의 이름, 날자 또는 감시기의 읽기값과 같은 단일값을 포함한다. 마당은 그의 길이와 자료형(실례로 ASCII 문자렬, 10진수)으로 특징지어 진다. 파일설계에 따라서 마당은 고정길이 또는 가변길이일수 있다. 후자의 경우에 마당은 흔히 2～3개의 보조마당 즉 기억될 실제값, 마당의 이름, 어떤 경우에는 마당길이로 구성된다. 다른 가변길이마당들의 경우에는 마당들사이에 특수한 경계기호들을 사용하여 마당길이를 지적한다.

레코드는 어떤 응용프로그람에 의하여 한개 단위로 취급될수 있는 관련마당들의 집합이다. 실례로 종업원레코드는 이름, 사회보안번호, 직종분류, 입직날자, 기타 등등과 같은 마당을 포함할수 있다. 설계에 따라서 다시 레코드는 고정길이 또는 가변길이일수 있다. 레코드는 자기마당의 일부가 가변길이이거나 마당의 수가 변한다면 가변길이로 된다. 후자의 경우 매개 마당은 보통 마당이름을 동반한다. 두 경우에 다 완전한 레코드는 보통 길이마당을 포함한다.

파일은 류사한 레코드들의 집합이다. 파일은 사용자들과 응용들에 의하여 단일실체로서 취급되며 이름으로 참조될수 있다. 파일은 유일한 파일이름을 가지며 생성되고 삭제될수 있다. 접근조종제한들은 보통 파일준위에서 적용된다. 즉 공유체계에서 사용자와 프로그람들은 전체 파일들에 접근하거나 접근하지 못한다. 더 정교한 체계들에서 이러한 조종들은 레코드준위 또는 지어 마당준위에서 수행된다.

자료기지는 관련자료들의 집합이다. 자료기지의 본질적인 특징은 자료요소들사이의 관계가 명백하고 수많은 다른 응용들에서 사용하기 위하여 설계된다는것이다. 자료기지는 사무기관 또는 과학연구프로젝트와 관련되는 모든 정보를 포함할수 있다. 자료기지그자체는 한개이상의 파일형태들로 구성된다. 비록 조작체계가 몇가지 파일관리프로그람들을 사용할수 있다고 하더라도 보통 조작체계와 독립인 개별적인 자료기지관리체계가 존재한다.

사용자들과 응용들은 파일들을 사용하려고 한다. 이를 위하여 지원하여야 할 대표적인 조작들은 다음과 같다[LIVAPO].

- **완전검색** : 파일의 모든 레코드들을 검색한다. 파일에 있는 모든 정보를 동시에 처리하여야 하는 응용들은 이 조작을 요구한다. 실례로 파일의 정보에 대한 개요를 작성하는 응용은 모든 레코드들을 검색할것을 요구한다. 이 조작은 모든 레코드들에 순차적으로 접근하기때문에 순차처리라는 용어와 일치한다.
- **하나검색** : 이것은 단일레코드의 검색을 요구한다. 서로 대화하는 트랜잭션지향응용들은 이 조작을 요구한다.
- **다음검색** : 이것은 어떤 론리적순서에서 가장 최근에 검색한 레코드의 《다음》에 있는 레코드에 대한 검색을 요구한다. 양식을 작성하는것과 같은 서로 대화하는 일부 응용들은 이러한 조작을 요구할수 있다. 검색을 수행하는 프로그람도 역시 이 조작을 사용할수 있다.
- **앞검색** : 다음 레코드검색과 류사하지만 현재 접근한 레코드의 《앞》에 있는 레코드에 대한 검색을 요구한다.
- **하나삽입** : 파일에 새로운 레코드를 삽입한다. 파일의 순차를 보존하기 위하여 새로운 레코드를 어떤 특정한 위치에 끼워 넣는것이 필요할수 있다.
- **하나삭제 : 현**재 레코드를 삭제한다. 파일의 순차를 보존하기 위하여 일정한 련결들 또는 다른 자료구조들을 갱신할 필요가 있다.
- **하나갱신** : 한개의 레코드를 검색하고 그 레코드의 한개이상의 마당을 갱신하며 갱신된 레코드를 파일에 다시 써 넣는다. 레코드의 길이가 변하였으면 레코드길이가 변화되지 않았을 때보다 갱신조작이 일반적으로 더 힘들어 진다.
- **다수검색** : 다수의 레코드를 검색한다. 실례로 응용들 또는 사용자들은 일정한 기준모임을 만족시키는 모든 레코드들을 검사하려고 할수 있다.

파일에 대하여 가장 일반적으로 수행되는 조작들의 특징은 제2절에서 론의되는바와 같이 파일을 조직하는 방법에 영향을 미친다.

파일관리체계

파일관리체계는 파일을 사용하는 사용자와 응용에 봉사하는 체계쏘프트웨어의 모임이다. 대표적으로 사용자와 응용이 파일에 접근할수 있는 유일한 방법은 파일관리체계를 사용하는것이다. 파일관리체계는 매개 응용에 대하여 전용쏘프트웨어를 개발하여야 할

요구로부터 사용자와 프로그람작성자를 해방하며 체계에 자기의 가장 중요한 자산을 조종할 수단을 준다. [GR086]은 파일관리체계에 다음과 같은 목표들을 제기한다.

- 자료의 기억과 앞에서 언급한 조작들을 수행할 능력을 포함하는 사용자의 자료관리 요구들을 만족시키는것
- 파일의 자료가 유효하다는것을 가능한 한도까지 보증하는것
- 체계의 관점에서는 전반적인 처리능력의 측면에서 그리고 사용자의 관점에서는 응답시간의 측면에서 량쪽 다 성능을 최량화하는것
- 다양한 기억장치형들에 대한 입출력을 지원하는것
- 자료가 손상되거나 또는 파괴될 가능성을 최소로 하거나 제거하는것
- 표준화된 입출력대면부루틴들의 모임을 보장하는것
- 다중사용자체계들의 경우에 다중사용자들을 위한 입출력지원을 보장하는것

사용자의 요구를 만족시키는 첫번째 목표에서 그 요구들의 범위는 응용들의 종류와 콤퓨터체계가 사용될 환경에 관계된다. 대화하는 범용체계에서는 다음과 같은것들이 최소요구모임을 구성한다.

1. 매 사용자는 파일을 생성, 삭제, 읽기, 변경할수 있다.
2. 매 사용자는 다른 사용자의 파일에 대한 접근을 조종할수 있다.
3. 매 사용자는 사용자의 파일에 어떤 형의 접근을 허용하겠는가를 조종할수 있다.
4. 매 사용자는 문제에 적합한 형식으로 사용자파일들을 재구축할수 있다.
5. 매 사용자는 파일들사이에서 자료를 이동시킬수 있다.
6. 매 사용자는 여벌파일을 만들고 사용자의 파일이 손상된 경우에 회복할수 있다.
7. 매 사용자는 기호이름을 사용하여 사용자의 파일에 접근할수 있다.

이 목표들과 요구사항들은 파일관리체계를 론의하는 과정에 계속 상기될것이다.

파일체계구성방식

파일관리를 파악하기 위한 한가지 방도는 그림12-1에서 제시된바와 같이 대표적인 쏘프트웨어구성에 대한 서술을 고찰하는것이다. 물론 서로 다른 체계들은 서로 다르게 구성될것이지만 이 구성은 매우 대표적이다. 가장 낮은 준위에 있는 **장치구동프로그람**들은 주변장치, 장치조종기들 또는 통로들과 직접 통신한다. 장치구동프로그람은 장치에 대한 입출력조작들을 시동하고 입출력요구처리를 완성한다. 파일조작을 할 때 조종하는 대표적인 장치는 디스크와 테프구동기이다. 장치구동프로그람들은 보통 조작체계의 부분으로 간주된다.

다음 준위는 **기본파일체계** 또는 **물리적입출력**준위이다. 이것은 콤퓨터체계를 제외하면 환경과의 1차대면부이다. 이 준위에서는 디스크 또는 테프체계와 교환되는 자료블로크들을 처리한다. 따라서 이 준위는 그 블로크들을 2차기억장치에 배치하고 주기억기에 완충하는 일을 수행한다. 기본파일체계는 자료의 내용 또는 사용되는 파일의 구조를 리해하지 않는다. 기본파일체계는 흔히 조작체계의 부분으로 간주된다.

기본입출력감독기는 모든 파일입출력을 개시하고 완료한다. 이 준위에서 조종구조들은 장치입출력, 일정작성, 파일상태들을 취급한다. 기본입출력감독기는 파일이 선택된 상

태에서 파일입출력을 집행하는 장치를 선택한다. 기본입출력감독기는 또한 성능을 최대로 발휘하기 위하여 디스크와 테프접근을 일정작성하는데 관계한다. 이 준위에서는 입출력완충기들이 할당되고 2차기억기가 배정된다. 기본입출력감독기는 조작체계의 부분이다.

론리입출력은 사용자와 응용들이 레코드에 접근할수 있도록 한다. 따라서 기본파일체계가 자료블로크를 처리한다면 론리입출력모듈은 파일레코드들을 처리한다. 론리입출력은 범용레코드입출력능력을 제공하며 파일들에 대한 기본자료를 유지한다.

사용자에게 가장 가까운 파일체계준위를 **접근방법**이라고 한다. 접근방법은 응용들과 파일체계, 자료를 유지하는 장치들사이의 표준대면부를 준다. 서로 다른 접근방법은 서로 다른 파일구조와 자료에 접근하고 처리하는 서로 다른 방법을 반영한다.

가장 공통적인 접근방법들을 그림 12-1에서 보여 주며 제2절에서 간단히 설명한다.

파일관리기능

파일체계의 기능을 고찰하는 또다른 방법을 그림 12-2에서 보여 준다. 이 선도를 왼쪽에서 오른쪽으로 따라 가 보자. 사용자들과 응용프로그람들은 파일들을 생성하고 삭제하는 지령들과 파일들에 대한 조작을 수행하는 지령들로 파일체계와 대화한다. 파일체계는 임의의 조작을 수행하기전에 선택된 파일을 확인하고 그의 위치를 알아 내야 한다. 이것은 모든 파일들의 위치와 그의 속성들을 서술하는 어떤 종류의 등록부를 사용할것을 요구한다. 게다가 대부분의 공유체계들은 사용자접근조종을 수행한다. 즉 오직 권한받은 사용자들만 특수한 방법으로 특정한 파일들에 접근하게 한다. 사용자와 응용이 파일에 대하여 수행할수 있는 기본조작들은 레코드준위에서 수행된다. 사용자 또는 응용은 파일을 순차구조와 같이 레코드를 구성하는 어떤 구조를 가지는것으로 간주한다(실례로 종업원레코드는 이름을 알파베트순서로 기억한다.). 따라서 사용자지령들을 특정파일조작지령들로 변환하자면 파일구조에 적합한 접근방법을 사용하여야 한다.

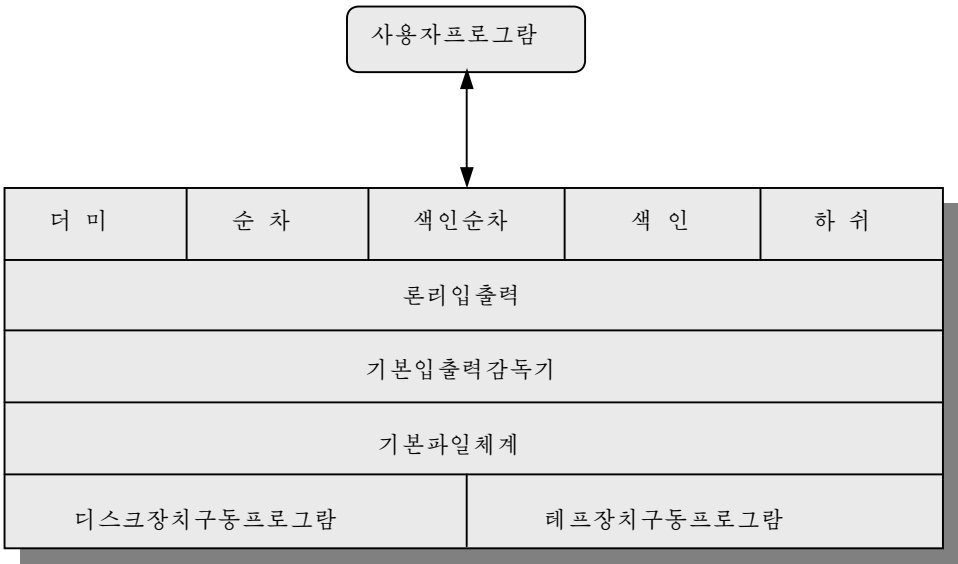



그림 12-1. 파일체계쏘프트웨어구성[GROS86]

사용자와 응용들은 레코드에 관계되며 입출력은 블로크를 기초로 하여 수행된다. 따라서 파일의 레코드들은 출력을 위하여 블로크화되고 입력을 위하여 비블로크화되여야 한다. 파일들에 대한 블로크입출력을 지원하자면 여러가지 기능들이 필요하다. 2차기억장치를 관리하여야 한다. 즉 2차기억기에 있는 자유블로크들에 파일들을 배정하고 어느 블로크들이 새로운 파일들과 현존 파일들의 성장에 유효한가를 알도록 자유기억기들을 관리하여야 한다. 게다가 개별적인 블로크입출력요구들은 일정작성되여야 한다. 이 문제는 제11장에서 취급되였다. 디스크일정작성과 파일배정은 다 성능최량화에 관계된다. 역시 이 기능들은 함께 고찰될것을 요구한다. 더우기 최량화는 파일의 구조와 접근패턴에 관계된다. 따라서 성능의 관점에서 최량파일관리체계를 개발하는것은 매우 복잡한 과제이다.

그림은 레코드처리를 하는 교차점에서 개별적인 체계유틸리티로서 간주되는 파일관리사건들과 조작체계사건들사이의 분할을 보여 준다. 이 분할은 임의로 한것이며 체계에 따라 여러가지 방법을 취할수 있다.

이 장의 나머지부분에서는 그림 12-2에서 제기된 몇가지 설계문제들을 고찰한다. 먼저 파일조직과 접근방법들을 론의한다. 이 문제는 비록 보통 고찰되는 조작체계의 범위를 벗어 나지만 파일조직과 접근에 대한 리해가 없이 다른 파일관련설계문제들을 평가하는것은 불가능하다. 다음에는 파일등록부의 개념을 고찰한다. 나머지 문제는 파일관리를 물리적입출력의 관점에서 취급하며 조작체계설계의 견지에서는 적당히 취급한다. 이러한 문제의 하나는 론리적레코드들을 물리적블로크들로 구성하는 방법이다. 결국 2차기억기에 파일을 배정하는 문제와 자유2차기억기의 관리문제가 제기된다.

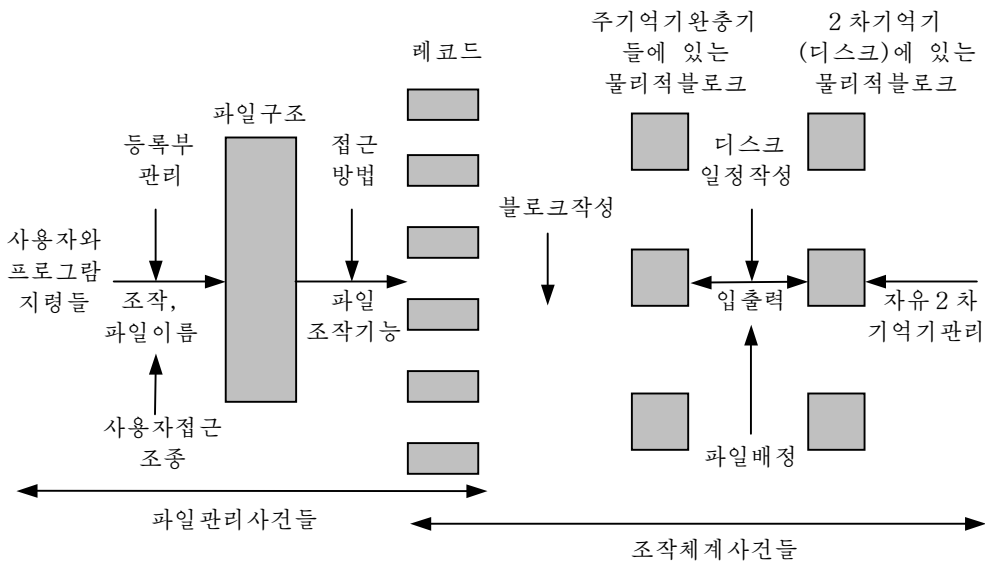



그림 12-2. 파일관리요소

제 2 절. 파일조직 및 접근

이 절에서는 레코드들에 접근하는 방법에 따라 결정되는 레코드들의 론리적구조화를 언급하기 위하여 파일조직이라는 용어를 사용한다. 2차기억기에 있는 파일의 물리적조직은 블로크작성전략과 파일배정전략 그리고 이 장에서 후에 취급하는 문제들에 관계된다.

파일조직을 선택하는데서 몇가지 중요한 기준이 있다.

- 신속한 접근
- 갱신의 쉬움
- 기억기의 경제성
- 간단한 보수
- 신뢰성

이 기준들의 상대적인 우선권은 파일들을 사용하는 응용들에 관계된다. 실례로 파일이 일괄방식으로만 처리되고 매번 모든 레코드들에 접근한다면 단일레코드를 검색하기 위한 신속한 접근은 그리 중요하지 않다. CD-ROM에 기억된 파일은 결코 갱신될수 없으며 따라서 갱신의 쉬움은 문제로 되지 않는다.

이 기준들은 충돌할수 있다. 실례로 기억기의 경제성때문에 자료의 여유도가 최소로 될것이다. 다른 한편 여유도는 자료에로의 접근속도를 증가시키는 1차적인 수단이다. 이러한 실례로 색인들의 사용을 들수 있다.

실현되였거나 제기된 파일조직방법의 수는 헤아릴수 없이 많으며 파일체계에 대하여 서술한 책도 많다. 이 책에서는 다섯가지 기본조직을 륜곽적으로 간단히 고찰한다. 실제로 사용되는 대부분의 구조들은 이 부류중 하나로 귀착되든가 이 조직들의 결합으로 실현될수 있다. 다섯가지 조직은 다음과 같다. 그중 첫 4개는 그림 12-3에서 보여 준다.

- 더미
- 순차파일
- 색인달린 순차파일
- 색인달린 파일
- 직접파일 또는 하쉬파일

표 12-1 은 이 다섯가지 조직의 상대적성능을 요약한다

표 12-1. 다섯가지 기본 파일조직의 성능[WIED87]

파일조직방법	공간		갱신		검색		
	속성들		레코드크기		단일 레코드	부분레코드 모임	완전 레코드
	가변	고정	같다	더 크다			
더미	A	B	A	E	E	D	B
순차파일	F	A	D	F	F	D	A
색인달린 순차파일	F	B	B	D	B	D	B
색인달린 파일	B	C	C	C	A	B	D
직접파일 또는 하쉬파일	F	B	B	F	B	F	E

A = 우수하다. 이 목적에 잘 맞는다. $O(r)$
B = 좋다. $O(o \times r)$
C = 적합하다. $O(r \log n)$
D = 어떤 특별한 노력이 필요하다. $O(n)$
E = 극단한 노력에 의하여 가능하다. $O(r \times n)$
F = 이 목적에 맞지 않는다. $O(n^{>1})$

여기서 r = 결과의 크기
o = 넘쳐 나는 레코드의 수
n = 파일에서 레코드의 수

더미

파일조직의 가장 덜 복잡한 형식을 더미라고 볼수 있다. 자료는 도착한 순서로 수집된다. 매개 레코드는 한개의 자료무리로 구성된다. 더미의 목적은 단순히 자료무리를 축적하고 그것을 보관하는것이다. 레코드들은 서로 다른 마당들 또는 서로 다른 순서의 류사한 마당들을 가질수 있다.

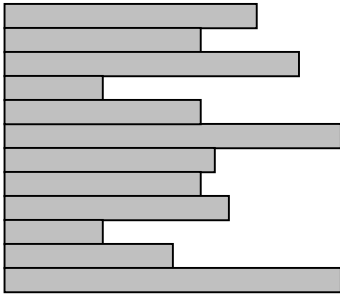
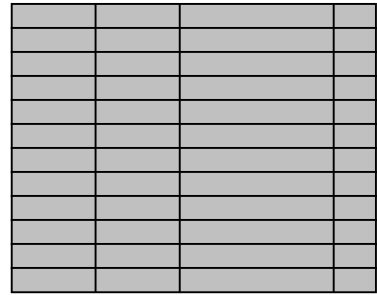
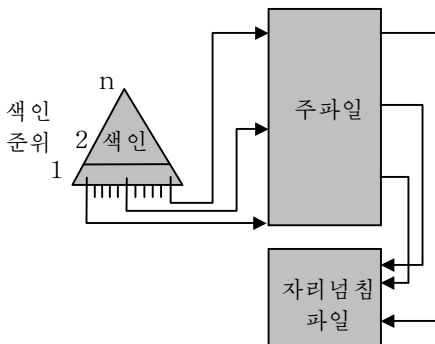
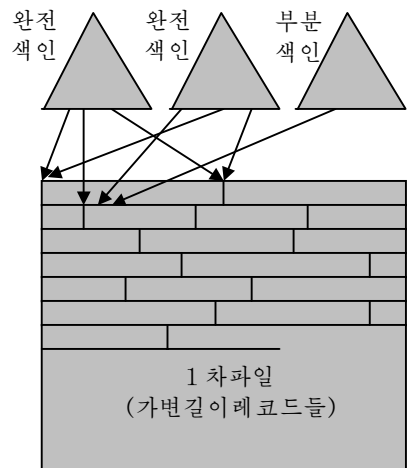



그림 12-3. 일반적인 파일조직방법
ㄱ-더미파일, ㄴ-순차파일, ㄷ-색인달린 순차파일, ㄹ- 색인달린 파일

따라서 매개 마당은 자기자체를 서술하며 값은 물론 마당이름도 포함한다. 매개 마당은 경계기호로 지적되여야 하며 보조마당으로서 명백히 포함되든가 그 마당형의 기정값으로서 알려 져야 한다.

더미파일들은 구조가 전혀 없기때문에 레코드접근은 완전탐색으로 진행된다. 즉 특정한 값을 가지는 특정한 마당을 포함하는 레코드를 찾으려고 하면 필요한 레코드를 찾을 때까지 또는 전체 파일을 탐색할 때까지 더미에 있는 매개 레코드를 조사해야 한다.

특정한 마당을 포함하는 또는 특정값을 가진 마당을 포함하는 모든 레코드를 찾으려면 파일전부를 탐색하여야 한다.

더미파일들은 처리에 앞서 자료를 수집하고 기억할 때 또는 자료를 조직하기 힘들 때 맞다 들게 된다. 이러한 형태의 파일은 기억된 자료의 크기와 구조가 변할 때 공간을 효과적으로 사용하며 완전탐색에 아주 적합하고 갱신하기가 쉽다. 그러나 이러한 제한된 사용을 제외하면 이런 형태의 파일은 대부분의 응용들에 적합하지 않다.

순차파일

파일구조의 가장 공통적인 형식은 순차파일이다. 이런 형태의 파일들에서는 고정형식의 레코드를 사용한다. 모든 레코드들은 같은 길이로 되여 있으며 특정한 순서로 된 같은 수의 고정길이마당들로 구성되여 있다. 매개 마당의 길이와 위치는 알려 져 있기때문에 마당들의 값만 기억되여야 한다. 매개 마당의 이름과 길이는 파일구조의 속성들이다.

한개의 특정한 마당, 보통 매개 레코드에서 첫번째 마당을 열쇠마당이라고 한다. **열쇠마당**은 레코드를 유일하게 식별한다. 따라서 서로 다른 레코드들에 대한 열쇠값들은 서로 다르다. 게다가 레코드들은 열쇠순서로 기억된다. 즉 본문열쇠는 알파베트순서로, 수자열쇠는 수자순서로 기억된다.

순차파일들은 대표적으로 일괄응용들에서 사용되며 일반적으로 모든 레코드들에 대한 처리를 요구하는 응용들(실례로 계산응용 또는 생활비지불응용)에 가장 적합하다. 순차파일조직은 디스크는 물론 테프에 쉽게 기억되는 유일한 파일조직이다.

개별적인 레코드들에 대한 질문과 갱신을 요구하는 대화하는 응용들에서 순차파일은 성능을 저하시킨다. 접근은 열쇠가 일치하는 레코드를 찾기 위하여 파일에 대한 순차탐색을 요구한다. 주기억기에 한꺼번에 파일전체 또는 파일의 많은 몫을 가져 올수 있으면 더 효률적으로 탐색할수 있다. 그럼에도 불구하고 큰 순차파일의 레코드에 접근하자면 상당한 처리와 지연이 요구된다. 파일에로의 추가는 문제를 발생한다. 대표적으로 순차파일은 레코드들의 단순한 순차배렬로 블로크안에 기억된다. 즉 테프 또는 디스크상의 파일의 물리적조직은 파일의 론리적조직과 직접 대응한다. 이 경우에 보통 절차는 운영일지파일 또는 트랜잭션파일이라는 개별적인 더미파일에 새로운 레코드들을 배치한다. 운영일지파일을 주파일과 융합하여 정확한 열쇠순차로 새로운 파일을 산생하는 일괄갱신이 주기적으로 수행된다.

다른 방법은 순차파일을 련결목록으로서 물리적으로 조직하는것이다. 한개 또는 그 이상의 레코드가 매개 물리적블로크에 기억된다. 디스크상의 매개 블로크는 다음 블로크에로의 지시자를 포함한다. 새로운 레코드의 삽입은 지시자조작을 요구하지만 이 새로운 레코드들은 특정한 물리적블로크위치를 차지하지 않아도 된다. 이때 얻어 지는 추가적인 편리로 하여 보충적인 처리를 해야 하고 간접소비시간은 그만큼 커진다.

색인달린 순차파일

순차파일의 결함을 극복하기 위한 보통 방법은 색인달린 순차파일이다. 색인달린 순차파일은 순차파일의 열쇠특성을 그대로 가진다. 즉 레코드들은 열쇠마당에 기초한 순차

로 조직된다. 두가지 특징이 추가되였다. 즉 임의의 접근을 지원하기 위한 파일에로의 색인과 넘침파일이 추가되였다. 색인은 필요한 레코드의 부근에로 빨리 도달할수 있는 탐색능력을 준다. 넘침파일은 순차파일과 함께 사용되는 운영일지파일과 류사하지만 넘침파일의 레코드들이 자기 앞의 레코드의 지시자가 가리키는곳에 놓이도록 통합된다.

가장 간단한 색인달린 순차구조에서는 단일준위색인화가 사용된다. 이 경우에 색인은 단순한 순차파일이다. 색인파일의 매개 레코드는 두개 마당 즉 기본파일에서 열쇠마당과 같은 열쇠마당과 기본파일에로의 지시자로 구성된다. 특정한 마당을 찾기 위하여 색인을 탐색하며 필요한 열쇠값과 같거나 보다 큰 가장 가까운 열쇠값을 찾는다. 탐색은 기본파일에서 지시자가 지적하는 위치로부터 계속된다.

이 방법의 효과성을 보기 위하여 백만개의 레코드로 된 순차파일을 고찰하자. 특정한 열쇠값을 탐색하자면 평균 50만번의 레코드접근이 필요하다. 기본파일에 골고루 분산된 열쇠를 가진 1000개 입구를 포함하는 색인을 구축한다고 가정하자. 레코드를 찾자면 기본파일에 500번 접근한 다음 색인파일에 평균 500번 접근하여야 할것이다. 이때 평균탐색길이는 500,000에서 1000으로 줄어 든다.

파일에로의 추가는 다음의 방법으로 조종된다. 즉 기본파일의 매개 레코드는 응용에서는 볼수 없는 넘침파일에로의 지시자인 추가마당을 포함한다. 새로운 레코드를 파일에 삽입하려고 할 때에는 넘침파일에 그것을 추가한다. 론리적인 순서에서 새로운 레코드의 바로 앞에 있는 기본파일의 레코드는 갱신되여 넘침파일에 있는 새로운 레코드에로의 지시자를 포함한다. 바로 앞에 있는 레코드가 넘침파일에 있는 레코드이면 그 레코드는 갱신된다. 순차파일의 경우와 같이 색인달린 순차파일은 일괄방식에서 넘침파일과 때때로 통합된다.

색인달린 순차파일은 파일의 순차적성질을 손상시킴이 없이 단일레코드의 접근에 필요한 시간을 크게 줄인다. 파일전체를 순차적으로 처리하기 위하여 넘침파일에로의 지시자를 발견할 때까지 주파일의 레코드들을 순차로 처리하며 넘침파일에로의 지시자를 발견하면 령지시자를 만날 때까지 넘침파일에서 접근을 계속한다. 령지시자를 만나면 주파일의 다음레코드에로 다시 돌아와 검색을 계속한다.

접근에서 더 큰 효률을 얻기 위하여 다중준위색인화를 사용할수 있다. 따라서 가장 낮은 준위의 색인파일은 순차파일로 취급되며 고준위색인파일은 그 파일을 위하여 생성된다. 백만개 레코드로 된 파일을 다시 고찰하자. 만개의 입구점을 가진 저준위색인이 구축된다. 그다음에 100개 입구로 된 저준위색인에로의 고준위색인이 구축된다. 탐색은 고준위색인에서부터 시작하여(평균길이=50번 접근) 저준위색인에로의 입구점을 찾는다. 그다음 저준위의 색인을 탐색하여(평균길이=50) 기본파일에로의 입구점을 찾으며 계속하여 기본파일을 탐색한다(평균길이=50). 따라서 평균탐색길이는 500,000으로부터 1000으로, 150으로 줄어 들었다.

색인달린 파일

색인달린 순차파일은 순차파일의 한가지 제한을 그대로 가지고 있다. 즉 효률적인 처리는 파일의 단일마당에 의거한다는 제한을 가지고 있다. 열쇠마당이 아니라 어떤 다른 속성에 기초하여 레코드를 검색할 필요가 있을 때 이 두가지 형식의 순차파일은 적당

하지 않다. 어떤 응용들에서는 이러한 유연성이 필요하다.

이 유연성을 달성하기 위하여 탐색주제가 될수 있는 매개 마당형에 대하여 하나씩 색인을 가지는 다중색인을 채용한 구조가 필요하다. 일반 색인파일에서의 순차성과 단일 열쇠의 개념은 없어 진다. 레코드들은 오직 자기의 색인들을 통하여서만 접근된다. 결과는 적어도 한개의 색인에 있는 지시자가 그 레코드를 가리키는한 레코드들의 배치에 아무런 제한도 없다는것이다. 게다가 가변길이레코드를 받아 들일수 있다.

두가지형의 색인이 사용된다. 완전색인은 기본파일의 매개 레코드에 대하여 한개의 입구를 포함한다. 색인 그자체는 탐색을 쉽게 하기 위하여 순차파일로 구성한다. 부분색인은 관심이 있는 마당들이 존재하는 레코드들에로의 입구들을 포함한다. 가변길이레코드의 경우에 어떤 레코드들은 모든 마당들을 포함하지 않을것이다. 새로운 레코드를 기본파일에 추가할 때 모든 색인파일들을 갱신하여야 한다.

색인달린 파일들은 대체로 정보의 시간요구가 정밀하고 자료가 드물게 완전처리되는 응용들에서 사용된다. 실례로 항공예약체계와 재고품조종체계들을 들수 있다.

직접파일 또는 하쉬파일

직접파일 또는 하쉬파일은 디스크의 능력을 활용하여 임의로 알려 진 주소의 블로크에 직접 접근할수 있는 파일이다. 순차 및 색인달린 순차파일의 경우와 같이 매개 레코드에 열쇠마당이 필요하다. 그러나 여기에서 순차배렬의 개념은 전혀 없다.

직접파일은 열쇠값에 대한 하쉬법을 사용한다. 이 기능은 부록 8-ㄱ에서 설명한다. 그림 8-26ㄴ에서는 하쉬파일에서 대표적으로 사용되고 있는 넘침파일을 가진 하쉬법구성형태를 보여 주고 있다.

직접파일들은 흔히 매우 신속한 접근을 요구하는곳에서, 고정길이레코드를 사용하는 곳에서 그리고 레코드에 항상 동시에 한번 접근하는곳에서 사용된다. 실례로 등록부들, 가격표들, 일정작성들, 이름목록들을 들수 있다.

제 3 절. 파일등록부

내용

임의의 파일관리체계와 파일들의 집합과 관련되는것은 파일등록부이다. 등록부는 속성, 위치, 소유권과 함께 파일들에 대한 정보를 포함한다. 특히 2차기억기와 관련되는 대부분의 정보는 조작체계가 관리한다. 등록부는 그자체가 파일로서 조작체계가 소유하며 여러가지 파일관리루틴들에 의하여 접근할수 있다. 비록 등록부의 몇가지 정보는 사용자와 응용들에서 사용될수 있지만 이것은 일반적으로 체계루틴이 간접적으로 제공한다. 따라서 사용자들은 읽기전용방식에서조차 등록부에 직접 접근할수 없다.

표 12-2는 체계에 있는 매개 파일에 관하여 등록부에 대표적으로 기억시킨 정보를 제시한다. 사용자의 관점에서 보면 등록부는 사용자와 응용에 알려 진 파일이름들과 파일 그자체들사이의 사영을 제공한다. 따라서 매개 파일입구는 파일의 이름을 포함한다. 실제상 모든 체계들은 서로 다른 형태의 파일들과 서로 다른 파일조직들을 취급하며 역시 이 정보를 제공한다. 매개 파일에서 중요한 부류의 정보는 그의 위치, 크기와 함께 그의 2차기억기에 관계된다. 공유체계들에서는 파일에로의 접근을 조종하는데 사용되는 정

보를 제공하는것도 역시 중요하다. 대표적으로 한 사용자는 파일의 소유자이며 다른 사용자들에게 일정한 접근특권을 줄수 있다. 결국 파일의 현재 사용을 관리하고 그의 사용경력을 등록하는 사용정보가 필요하다.

구조

표 12-2 의 정보를 보관하는 방법은 여러가지 체계들에서 크게 다르다. 몇가지 정보는 파일과 관련된 머리부레코드에 기억될수 있다. 이것으로 하여 등록부에 필요한 기억기량이 줄어 들며 주기억기에 등록부의 모두 또는 대부분을 보다 쉽게 유지하여 속도를 개선한다. 물론 몇가지 주요한 요소들은 등록부에 있어야 한다. 대표적으로 이 주요한 요소들에는 이름, 주소, 크기, 조직방법이 포함된다.

등록부의 가장 단순한 형식의 구조는 매개 파일에 대한 입구목록으로 된 구조이다. 이 구조는 열쇠로 봉사하는 파일이름을 가진 단순한 순차파일로 표현될수 있다. 종전의 일부 단일사용자체계들에서 이 수법을 사용하여 왔다. 그러나 이 수법은 다중사용자들이 체계를 공유할 때 그리고 단일사용자들이 많은 파일들을 가질 때에는 적합하지 않다.

표 12-2. 파일등록부의 정보요소들

	기본정보
파일이름	생성자(사용자 또는 프로그람)가 선택하는 이름. 특정등록부내에서 유일하여야 한다.
파일형태	실례로 본문, 2 진, 적재모듈 기타 등등
파일조직	서로다른 조직방법들을 지원하는 체계들에 대하여
	주소정보
기록권	파일이 기억되는 장치를 가리킨다.
시작주소	2 차기억기의 물리적시작주소(실례로 디스크의 시린더, 자리길, 블로크번호)
사용된 크기	바이트, 단어블로크단위의 파일의 현재크기
배정된 크기	최대파일크기
	접근조종정보
소유자	이 파일에 대한 조종권이 배당되는 사용자, 소유자는 다른 사용자들에게 접근권을 허가/거절할수 있으며 이 특권들을 변경할수 있다.
접근정보	이 요소의 간단한 판본은 권한받은 매 사용자에 대하여 사용자이름과 통과암호를 포함한다.
허가된 작용	읽기, 쓰기, 집행, 망을 통한 전송을 조종한다.
	사용정보
생성된 날자	파일이 처음으로 등록부에 놓여 진 시간
생성자의 신원	보통은 현재소유자이지만 반드시 그렇지는 않다.
마지막읽기접근날자	레코드를 읽기한 마지막 시간의 날자
마지막읽기자의 신원	읽기를 한 사용자
마지막변경날자	마지막갱신, 삽입, 삭제날자
마지막변경자의 신원	변경을 한 사용자
마지막여벌작성날자	파일을 마지막으로 다른 2 차기억매체에 여벌로 만들 날자
현재사용	파일을 열고 있는 프로세스 또는 프로쎄스들과 같이 파일이 프로쎄스에 의하여 폐쇄되여 있는지 그리고 파일이 디스크에서가 아니라 주기억기에서 갱신되였는지 등 파일에 대한 현재동작정보

파일구조에 대한 요구들을 리해하기 위해서는 등록부에서 수행할수 있는 조작의 형태들을 고찰하는것이 유익하다.

- **탐색 :** 사용자 또는 응용이 파일을 참조할 때 등록부를 탐색하여 그 파일에 대응하는 입구를 찾아야 한다.
- **파일생성 :** 새로운 파일을 생성할 때 등록부에 입구를 첨부하여야 한다.
- **파일삭제 :** 파일을 삭제할 때 등록부에서 입구를 제거하여야 한다.
- **등록부목록 :** 등록부전체 또는 일부가 요구될수 있다. 일반적으로 이 요구는 사용자가 제기하며 사용자가 소유한 모든 파일들의 목록을 매개 파일의 일부 속성(실례로 형태, 접근조종정보, 사용정보)과 함께 제시한다.
- **등록부갱신 :** 일부 파일속성들이 등록부에 기억되기때문에 이 속성들중 하나가 변하면 대응하는 등록부입구를 변화시켜야 한다.

간단한 목록은 이 조작들을 지원하는데 적합하지 않다. 단일사용자의 요구를 고찰하자. 사용자는 문서처리본문파일, 도형파일, 표계산자료, 기타 등을 포함하여 많은 파일형태들을 가질수 있다. 사용자는 이 파일들을 프로젝트에 따라, 형태에 따라 또는 어떤 다른 편리한 방법으로 조직하려고 한다. 등록부가 간단한 순차목록이라면 등록부는 파일들을 조직하는데 그 어떤 도움도 주지 않으며 사용자가 두개의 서로 다른 형태의 파일에 같은 이름을 사용하지 않도록 한다. 문제는 공유체계에서 더욱 악화된다. 단일이름짓기는 심중한 문제를 일으킨다. 게다가 등록부에 자기 본래의 구조가 전혀 없으면 사용자들에에게 전체등록부의 일부분을 숨기기 힘들다.

이 문제들을 풀기 위한 출발점은 2준위기구를 사용하는것이다. 이 경우 매개 사용자에게는 한개 등록부와 주등록부가 있다. 주등록부는 매 사용자등록부에로의 입구를 가지며 주소와 접근조종정보를 제공한다. 매개 사용자등록부는 그 사용자파일들의 간단한 목록이다. 이러한 배치는 이름이 단일사용자의 파일집합내에서만 유일하여야 하며 파일체계가 등록부들에 대한 접근제한을 쉽게 수행할수 있다는것을 의미한다. 그러나 그것은 여전히 사용자들이 파일들의 집합을 구조화하는데 아무런 도움도 주지 못한다.

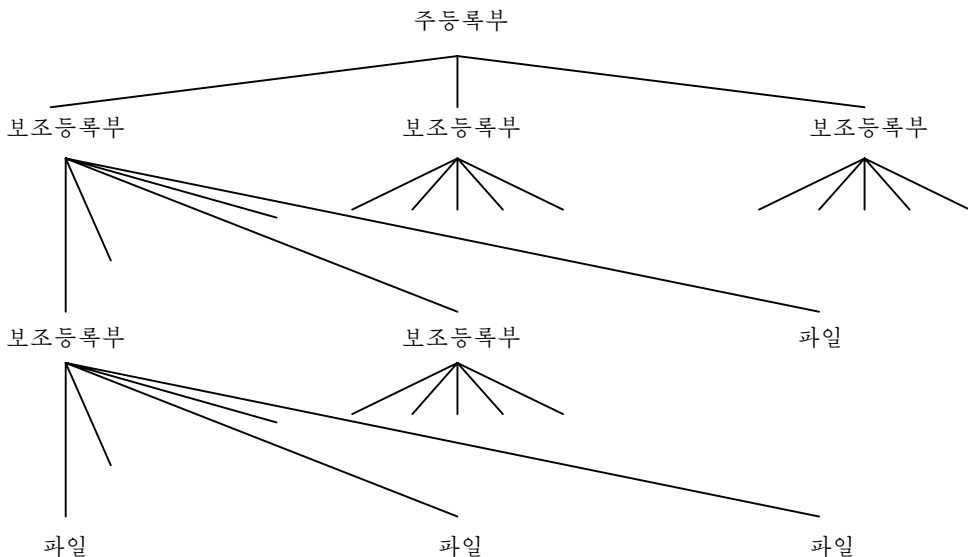



그림 12-4. 나무구조등록부

더 강력하고 유연한 방법이며 거의 보편적으로 도입되고 있는 방법은 계층 또는 나무구조방법이다(그림 12-4). 앞에서와 같이 자기 밑에 많은 사용자등록부를 가진 주등록부가 있다. 매개 사용자등록부들은 계속하여 입구들로서 보조등록부들과 파일들을 가질수 있다. 이것은 임의의 준위에 다 해당된다. 즉 임의의 준위에서 등록부는 보조등록부에 대한 입구들과 파일들에 대한 입구들로 구성될수 있다.

마지막으로 등록부와 보조등록부를 어떻게 구성하는가가 문제이다. 물론 가장 단순한 방법은 매개 등록부를 순차파일로서 기억하는것이다. 등록부들이 매우 많은 입구들을 포함한다면 이러한 구성은 불필요하게 오랜 탐색시간을 초래할수 있다. 그런 경우에는 하쉬구조가 더 좋다.

이름짓기

사용자는 파일을 기호이름으로 지정하려고 한다. 체계에 있는 매개 파일은 파일참조를 명백히 하기 위하여 유일한 이름을 가져야 한다. 다른 한편 특히 공유체계에서 사용자들에게 유일한 이름을 제시할것을 요구하는것은 사용자들이 받아 들일수 없는것이다.

나무구조등록부를 사용하면 유일한 이름을 할당하는데서 난관을 극복할수 있다. 파일에 도달할 때까지 뿌리 또는 주등록부로부터 여러 갈래에로의 경로를 추적하여 체계에 있는 임의의 파일의 위치를 찾을수 있다. 그자체가 파일이름으로 끝나는 등록부이름들의 렬은 그 파일에 대한 **경로명**을 구성한다. 실례로 그림12-5의 왼쪽 아래 모서리에 있는 파일은 경로명/User B/Word/Unit A/ABC를 가진다. 사선은 이름들을 차례로 구분하기 위하여 사용된다. 주등록부의 이름은 암시적이다. 왜냐하면 모든 경로는 그 등록부에서 시작하기때문이다. 파일들이 유일한 경로명을 가지는한 여러가지 파일들이 같은 파일이름을 가질수 있다. 따라서 체계에 파일이름이 ABC인 또다른 파일이 있지만 그것은 경로명/User B/Draw /ABC를 가진다.

경로명은 파일이름들의 선택을 쉽게 하지만 사용자가 파일에 대한 참조를 할 때마다 완전한 경로명을 한자한자 전부 쓴다는것은 불편할것이다. 대표적으로 서로 대화하는 사용자 또는 프로쎄스가 흔히 **작업등록부**라고 하는 현재 등록부를 경로명에 련관시켰다. 그때 파일들은 작업등록부에 상대적으로 참조된다. 실례로 사용자 B의 작업등록부가 《Word》이면 경로명 Unit A/ABC는 그림 12-5의 왼쪽 아래 모서리에 있는 파일을 식별하는데 충분하다. 서로 대화하는 사용자가 등록할 때 또는 프로쎄스가 생성될 때 작업등록부에 대한 기정값은 사용자등록부이다. 집행기간에 사용자는 뿌리등록부에서 아래우로 이동하면서 서로 다른 작업등록부들을 정의할수 있다.

제 4 절. 파일공유

다중사용자체계에서는 많은 사용자들속에서 파일들을 공유할데 대한 요구가 거의 항상 제기된다. 두가지 문제 즉 접근권과 동시접근관리가 제기된다.

접근권

파일체계는 사용자들사이에서 다방면적인 파일공유를 허용하는 유연한 도구를 보장하여야 한다. 파일체계는 특정한 파일에 접근하는 방법을 조종할수 있도록 여러가지 수법을 제공하여야 한다. 대표적으로 사용자들 또는 사용자들의 그룹들에는 파일에 대한 일정한 접근권이 주어 진다. 넓은 령역의 접근권들이 사용되였다. 다음의 목록은 특정한

파일에 대하여 특정한 사용자에게 할당할수 있는 접근권을 나타낸다.

- **무효** : 사용자는 파일의 존재조차도 알수 없으며 더군다나 평가할수 없다. 이 제한을 실시하면 사용자는 이 파일을 포함하는 사용자등록부를 읽을수 없다.
- **확인** : 사용자는 파일이 존재하는가, 소유자가 누구인가를 확인할수 있다. 사용자는 소유자에게 추가적인 접근권들을 요구할수 있다
- **집행** : 사용자는 프로그람을 적재하고 집행할수 있지만 복사할수 없다. 독점프로그람들은 흔히 이 제한으로 접근가능하다.

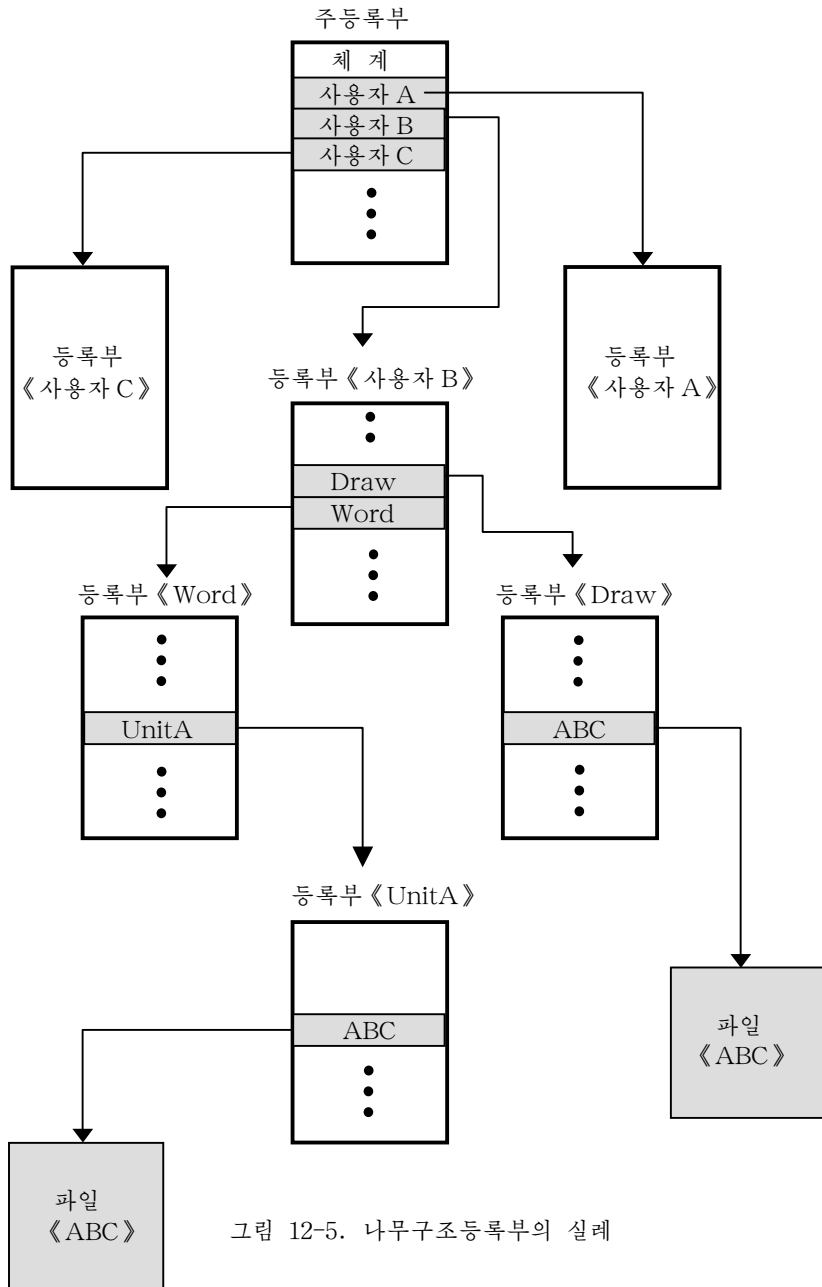



그림 12-5. 나무구조등록부의 실례

- **읽기** : 사용자는 복사와 집행을 포함하여 임의의 목적으로 파일을 읽을수 있다. 어떤 체계들은 보기와 복사를 구별할수 있다. 전자의 경우에 파일의 내용을 사용자에게 현시할수 있지만 사용자는 복사할 그 어떤 수단도 없다.
- **추가** : 사용자는 흔히 파일의 끝에 자료를 추가할수 있지만 파일의 내용을 변경하거나 지울수 없다. 이 권한은 많은 자원에서 자료를 수집할 때 유익하다.
- **갱신** : 사용자는 파일의 자료를 변경, 삭제, 추가할수 있다. 이것은 일반적으로 파일에 초기쓰기, 완전 또는 부분적인 재쓰기 그리고 자료의 전부 또는 한부분의 삭제를 포함한다. 일부 체계들은 서로 다른 갱신정도를 구별한다.
- **보호변경** : 사용자는 다른 사용자들에게 부여된 접근권들을 변경시킬수 있다. 대표적으로 파일소유자만이 이 권한을 가진다. 일부 체계들에서 소유자는 다른 사용자들에게 이 권한을 줄수 있다. 이 기구의 람용을 방지하기 위하여 파일소유자는 대표적으로 이 권한의 유지자가 어느 권한들을 변경시킬수 있는가를 규정할수 있다.
- **삭제** : 사용자는 파일체계에서 파일을 삭제할수 있다.

이 권한들은 매개 권한이 자기보다 앞에 있는 권한들을 내포하는 계층구조를 구성한다고 볼수 있다. 따라서 특정한 사용자에게 특정한 파일에 대한 갱신권이 부여되면 그때 그 사용자에게는 역시 다음의 권한 즉 확인, 집행, 읽기, 추가권도 부여된다.

보통 초기에 파일을 생성한 한명의 사용자가 주어 진 파일의 소유자로 지정된다. 소유자는 앞에서 렬거한 모든 접근권들을 가지며 다른 사용자들에게 권한들을 줄수 있다. 다음의 서로 다른 사용자부류들에 접근권을 줄수 있다.

- **특정사용자** : 사용자 ID 로 지정된 개별적인 사용자들
- **사용자그룹** : 개별적으로 정의되지 않는 사용자들의 모임. 체계는 사용자그룹의 성원임을 추적하는 어떤 방법을 가지고 있어야 한다.
- **전부** : 이 체계에로의 접근을 가지는 모든 사용자들. 이것들은 공개파일들이다.

동시접근

둘이상의 사용자에게 파일을 추가 또는 갱신하기 위한 접근을 허용할때 조작체계 또는 파일관리체계는 어떤 규칙을 세워야 한다. 억지방법은 사용자가 파일을 갱신하려고 할 때 파일전부를 봉쇄하도록 한다. 이것은 본질적으로 제5장에서 론의된 읽기자/쓰기자 문제이다. 호상배제와 교착문제들은 공유접근능력을 설계하는데서 강조되여야 한다.

제 5 절. 레코드의 블로크화

그림 12-2 에서 지적된바와 같이 레코드는 파일접근을 위한 론리적단위이며 반면에 블로크는 2 차기억기와의 입출력단위이다. 입출력을 수행할 때 레코드들을 블로크로 구성하여야 한다.

고려하여야 할 여러가지 문제들이 제기된다. 먼저 블로크들이 고정길이인가 또는 가변길이인가 하는것이다. 대부분의 체계에서 블로크들은 고정길이로 되여 있다. 이것은 입출력, 주기억기에서 완충기배정, 2차기억기에서 블로크들의 구성을 단순화한다. 다음으로 무엇이 블로크의 상대적인 크기와 평균레코드크기를 비교하는가 하는것이다. 교환조건은 다음과 같다. 즉 블로크가 크면 클수록 한번의 입출력조작으로 통과되는 레코드는 더 많아 진다. 파일이 순차적으로 처리되거나 탐색되면 리롭다. 왜냐하면 보다 큰 블로크를 사

용하여 입출력조작의 수를 줄이고 따라서 처리속도를 높일수 있기때문이다. 다른한편 레코드가 임의로 접근되고 특정한 참조의 국소성을 전혀 알수 없으면 보다 큰 블로크는 사용하지 않는 레코드들의 불필요한 이송을 초래한다. 그러나 순차조작의 빈도를 참조의 국소성에 대한 가능성과 결합하면 보다 큰 블로크를 사용하는것으로 입출력이송시간이 줄어 든다고 말할수 있다. 문제는 블로크가 클수록 더 큰 입출력완충기를 요구하며 완충기관리를 더 어렵게 한다는것이다.

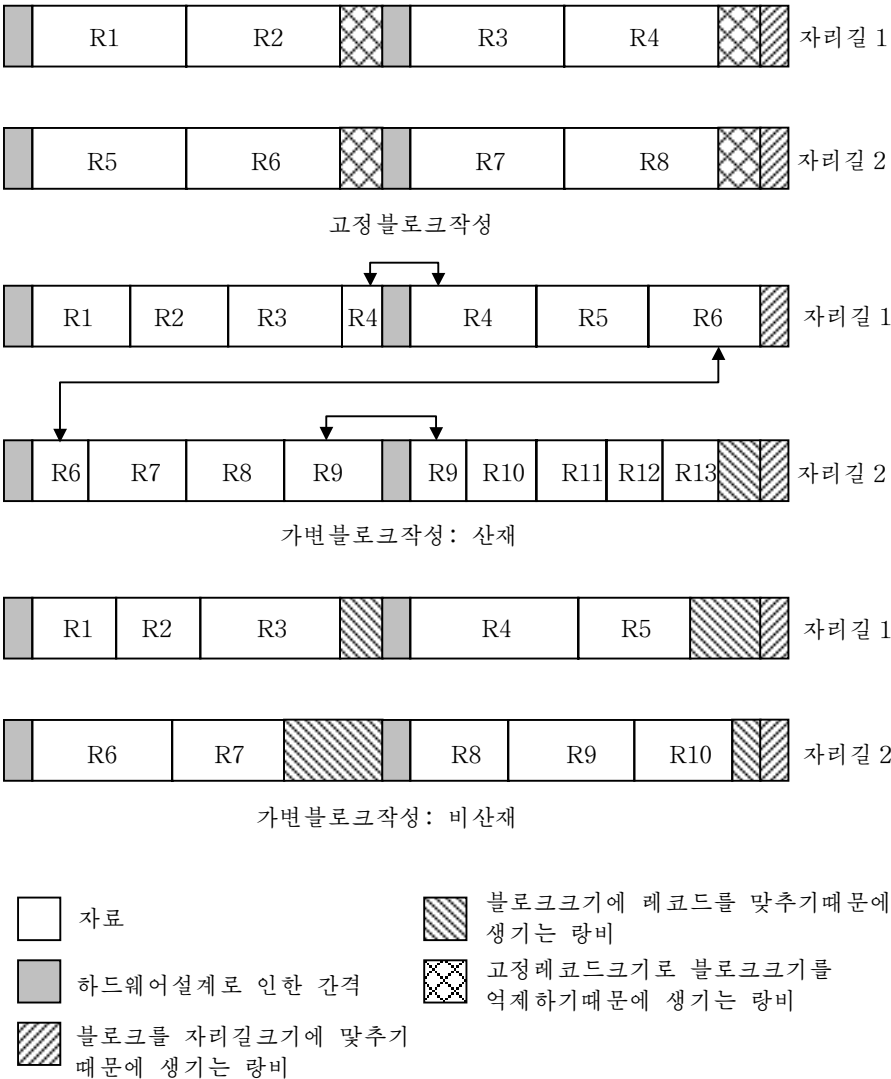



그림 12-6. 레코드블로크작성방법[WIED87]

블로크의 크기가 주어 지면 3가지 블로크작성방법을 사용할수 있다.

- **고정블로크작성** : 고정길이레코드를 사용하며 레코드의 전체수를 블로크에 기억한다. 매 블로크의 끝에 사용하지 않는 공간이 있을수 있다. 이것을 내부쪼각이라고 한다.

- **가변길이산재블로크작성** : 가변길이레코드를 사용하며 사용하지 않는 공간이 전혀 없게 블로크에 가변길이레코드를 채워 놓는다. 따라서 일부 레코드들은 계승블로크에로의 지시자가 지정하는 연장부분을 가지고 두개 블로크에 나뉘여 져야 한다.
- **가변길이비산재블로크작성** : 가변길이레코드를 사용하지만 산재기능을 사용하지 않는다. 다음 레코드가 나머지 사용하지 않는 공간보다 더 크면 블로크의 나머지를 사용할수 없기때문에 대부분 블로크들에 랑비되는 공간이 있다.

그림 12-6 은 디스크에 있는 순차적인 블로크들에 파일을 보관한다고 가정하는 방법들을 설명한다. 어떤 다른 파일배정기구를 사용한다고 해도 효과는 달라 지지 않을것이다(제 6 절을 보시오.).

고정블로크작성은 고정길이레코드를 가진 순차파일에서 공통방식이다. 가변길이산재블로크작성은 기억장치에서 효률적이며 레코드크기를 제한하지 않는다. 그러나 이 수법은 실현하기 힘들다. 두개 블로크에 산재해 있는 레코드들은 두번의 입출력조작을 요구하며 파일들은 그 구성을 고려하지 않고는 갱신하기가 힘들다. 가변길이비산재블로크작성은 공간을 랑비하며 레코드크기를 블로크크기로 제한한다.

레코드블로크작성수법은 가상기억기하드웨어를 받아 들였으면 하드웨어와 대화할수 있다. 가상기억기환경에서는 페지를 기본이송단위로 할것을 바란다. 페지는 일반적으로 아주 작다. 그래서 페지를 비산재블로크작성을 위한 블로크로 취급하는것은 비실용적이다. 따라서 일부 체계들은 다중페지들을 결합하여 파일입출력을 위한 더 큰 블로크를 창조한다. 이 방법은 IBM 대형콤퓨터에서 VSAM 파일에 사용된다.

제 6 절. 2 차기억기의 관리

2 차기억기에 있는 파일은 블로크들의 집합으로 구성된다. 조작체계나 파일관리체계는 파일들에 블로크들을 배정하여야 한다. 이것은 두가지 관리문제를 발생시킨다. 첫째로, 2 차기억기의 공간을 파일들에 배정하여야 하며 둘째로, 배정에 사용할 공간을 추적하는것이다. 이 두 과제는 서로 련관된다는것 즉 파일배정을 위한 방법이 자유공간관리를 위한 방법에 영향을 미칠수 있다는것을 알수 있다.

이 절에서는 먼저 단일디스크에 파일을 배정하는 방법들을 고찰한다. 그다음 자유공간관리문제를 고찰하며 마지막에 신뢰성을 론의한다.

파일배정

파일배정에서는 여러가지 문제들이 제기된다.

1. 새로운 파일을 생성할 때 파일에 필요한 최대공간이 즉시에 배정되는가?
2. 공간은 쪼각이라고 하는 한개이상의 련속적인 단위들로 파일에 배정된다. 한쪼각의 크기는 단일블로크크기로부터 전체 파일크기까지의 범위에 있을수 있다. 얼마만한 크기의 쪼각이 파일배정에 사용되는가?
3. 파일에 할당된 쪼각들을 추적하는데 어떤 종류의 자료구조 또는 표가 사용되는가? 이러한 표를 대표적으로 **파일배치표**(FAT)라고 한다.

이 문제들을 차례로 고찰하자.

미리배정 대 동적배정

미리배정방책은 파일생성을 요청하는 시각에 파일의 최대크기를 선언할것을 요구한

다. 프로그람번역, 요약자료파일들의 산생 또는 다른 체계로부터 통신망을 통한 파일의 전송과 같은 많은 경우에 이 값을 정확히 판단할수 있다. 그러나 많은 응용들에서 파일의 최대가능한 크기를 정확히 판단하는것은 힘들기는 하지만 가능하다. 그런 경우에 사용자들과 응용프로그람작성자들은 기억공간이 남도록 파일크기를 과대평가하기 쉽다. 이것은 2 차기억기배정의 관점에서 보면 명백히 랑비로 된다. 따라서 필요에 따라 파일에 쪼각으로 공간을 할당하는 동적배정을 사용하는것이 유익하다.

쪼각크기

두번째 문제는 파일에 배정되는 쪼각크기에 대한 문제이다. 한 극단에서 전체파일을 유지하는데 충분히 큰 쪼각을 배정한다. 다른 극단에서 한번에 한개 블로크씩 디스크공간을 배정한다. 쪼각의 크기를 선택하는데서 단일파일의 관점에서 본 효률과 전반체계의 효률사이에 교환조건이 있다. [WIED87]은 교환조건에서 고려할 4 가지 항목을 제시한다.

1. 련속공간은 특히 다음레코드검색조작에 대한 성능을 증가시키며 트랜잭션지향조작체계에서 실행하는 트랜잭션에서 성능을 크게 높인다.
2. 많은 수의 작은 쪼각들을 사용하면 배정정보를 관리하는데 필요한 표들의 크기가 커진다.
3. 고정크기쪼각들(실례로 블로크들)을 사용하면 공간의 재배정을 간단화한다.
4. 가변크기쪼각 또는 작은 고정크기쪼각들을 사용하면 과배정때문에 사용하지 않는 기억기의 랑비를 최소로 한다.

물론 이 항목들은 호상 작용하며 함께 고려하여야 한다. 그 결과 두가지 주요한 방안이 제기된다.

- **큰 가변련속쪼각** : 이것은 더 좋은 성능을 제공한다. 가변크기는 랑비를 없애며 파일배정표들은 작아 진다. 그러나 공간을 재사용하기는 힘들다.
- **블로크** : 작은 고정크기쪼각들은 더 큰 유연성을 제공한다. 그것들은 자기들의 배정에 큰 표들 또는 복잡한 구조를 요구할수 있다. 린접은 포기되였으며 블로크들은 요구에 따라 배정된다.

어느 방안이든 미리배정이나 동적배정에 적합하다. 큰 가변련속쪼각들의 경우에 파일은 하나의 련속블로크들의 그룹에 미리 배정된다. 이것은 파일배정표를 요구하지 않는다. 요구되는것은 첫 블로크에로의 지시자와 배정된 블로크의 수뿐이다. 블로크들인 경우에 요구된 모든 쪼각들은 한번에 배정된다. 이것은 파일에 대한 파일배정표가 여전히 고정크기일것이라는것을 의미한다.

가변크기쪼각인 경우에 자유공간의 분렬문제가 제기된다. 이 문제는 제 7 장에서 분할된 주기억기를 고찰할 때 론의했다. 몇가지 가능한 전략들은 다음과 같다.

- **첫블로크배정** : 자유블로크목록에서 사용하지 않는 충분한 크기의 련속블로크들의 첫 그룹을 선택한다.
- **최적블로크배정** : 충분한 크기의 사용하지 않는 가장 작은 그룹을 선택한다.
- **가장 가까운 배정** : 국소성을 확대하기 위하여 파일의 이전 배정과 가장 가까운 위치에 있는 충분한 크기의 사용하지 않는 그룹을 선택한다.

어느 전략이 가장 좋은가는 명백하지 않다. 여러가지 전략들을 모형화하는데서 난관은 파일의 형태, 파일의 접근패턴, 다중프로그람처리등급, 체계의 다른 성능요소들, 디스크고속완충, 디스크일정작성 기타 등을 포함하여 많은 요소들이 호상 작용한다는데 있다.

파일배정방법

미리배정 대 동적배정과 쪼각크기에 대한 문제들을 고찰한다음에는 특정한 파일배정방법들을 고찰할수 있다. 이 세가지 방법 즉 린접, 사슬, 색인방법들이 공통으로 사용된다. 표 12-3 은 매 방법의 특징들을 요약한다.

린접배정인 경우에는 파일생성시에 린접하는 블로크들의 단일모임을 파일에 배정한다(그림 12-7). 따라서 이것은 미리배정전략이며 가변크기쪼각들을 사용한다. 파일배정표는 시작블로크와 파일의 길이를 보여 주는 매개 파일에 대한 단일입구를 요구한다. 린접배정은 개별적인 순차파일의 견지에서 가장 좋은 방식이다. 한번에 다중블로크들을 줄수 있기때문에 순차처리에서 입출력성능을 개선한다. 또한 단일블로크를 검색하기도 쉽다. 실례로 파일은 블로크 b에서 시작하고 파일의 i번째 블로크가 요구된다면 2차기억기에서 그의 위치는 단지 $b+i-1$이다. 린접배정은 몇가지 문제점들을 제기한다. 외부분렬이 발생하여 충분한 길이를 가지는 린접블로크공간을 찾기 힘들것이다. 때때로 압축알고리듬을 수행하여 디스크에 추가적인 공간을 확보할 필요가 있다(그림 12-8). 또한 미리배정의 경우에도 앞에서 언급한 문제들과 함께 생성시에 파일의 크기를 선언할 필요가 있다.

파일배정표

파일이름	시작블로크	길이
파일 A	2	3
파일 B	9	5
파일 C	18	8
파일 D	30	2
파일 E	26	3

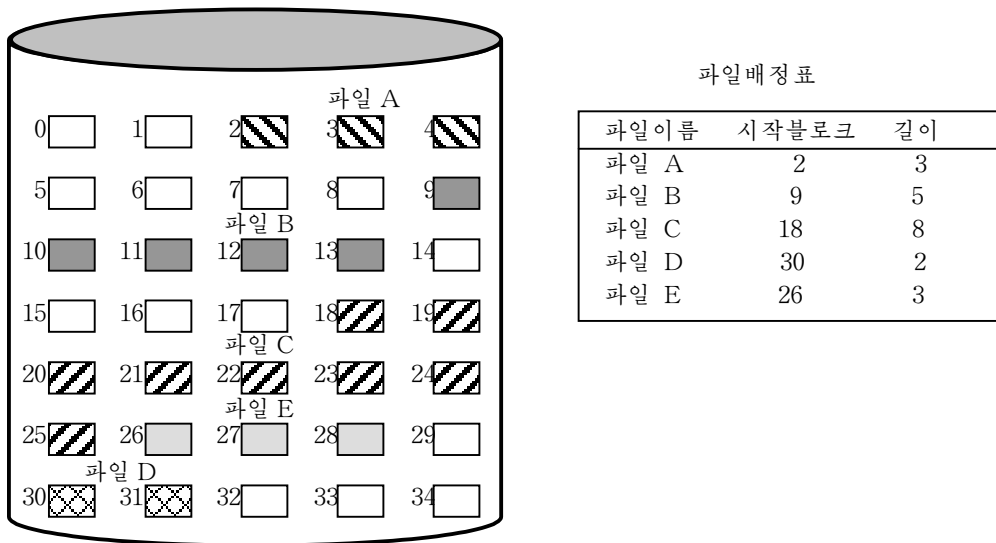



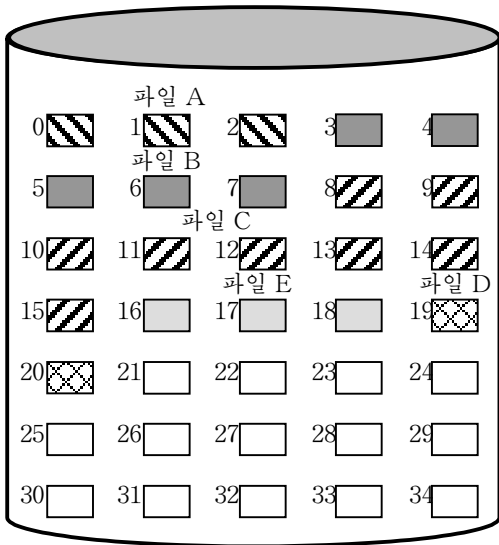
그림 12-7. 린접파일배정

표 12-3. 파일배정방법

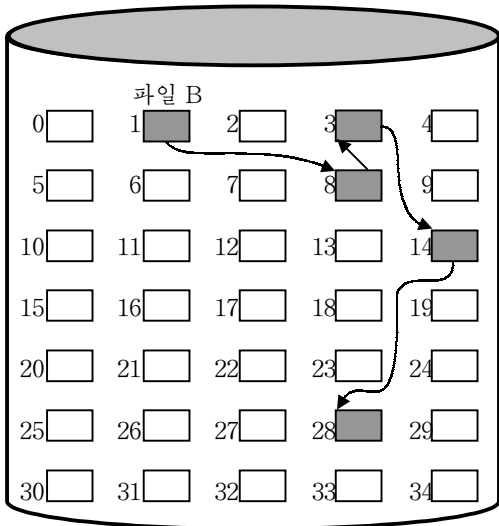
	린접	**사슬식**	**색인**	
미리배정	필수	가능	가능	
고정 또는 가변크기쪼각들	가변	고정블로크들	고정블로크들	가변
쪼각크기	크다	작다	작다	중간
배정빈도	한번	낮은데서 높은데로	높다	낮다
배정시간	중간	길다	짧다	중간
파일배정표크기	한개입구	한개입구	크다	중간

린접배정의 반대극단에는 **사슬식배정**이 있다(그림 12-9). 대표적으로 배정은 개별적인 블로크에 기초한다. 매개 블로크는 사슬에서 다음 블로크에로의 지시자를 포함한다. 파일배정표는 또 시작블로크와 파일의 길이를 보여 주는 매개 파일에로의 단일입구를 요구한다. 비록 미리배정이 가능할지라도 필요에 따라 블로크를 배정하는것이 아주 일반적이다. 블로크들의 선택은 쉬운 문제이다. 즉 임의의 자유블로크를 사슬에 추가할수 있다.

한번에 오직 하나의 블로크를 요구하기때문에 외부분렬이 전혀 발생하지 않는다. 이러한 형태의 물리적구성은 순차적으로 처리하려는 순차파일들에 가장 적합하다. 파일의 개별적인 블로크를 선택하기 위해서는 사슬전체에 대하여 필요한 블로크를 추적하여야 한다.

지금까지 서술한바와 같이 사슬식배정의 한가지 문제점은 국소성의 원리를 전혀 받아들일수 없다는것이다. 따라서 순차처리에서와 같이 한번에 여러개의 블로크들을 제공할 필요가 있다면 디스크의 서로 다른 부분에로의 련속적인 접근이 요구된다. 이것은 아마 단일사용자체계에 더 큰 영향을 주지만 공유체계에도 역시 련관될수 있다. 이 문제를 극복하기 위하여 일부 체계들은 파일들을 주기적으로 통합정리한다(그림 12-10).

파일배정표

파일이름	시작블로크	길이
파일 A	0	3
파일 B	3	5
파일 C	8	8
파일 D	18	2
파일 E	16	3

그림 12-8. 린접파일배정 (압축후에)

파일배정표

파일이름	시작블로크	길이
• • •	• • •	• • •
파일 B	1	5
• • •	• • •	• • •

그림 12-9. 사슬배정

색인에 의한 배정은 린접배정과 사슬식배정의 많은 문제들을 강조하여 취급한다. 이 경우에 파일배정표는 매개 파일에 대하여 개별적인 1 준위색인을 포함하는데 그 색인은 파일에 배정된 매개 쪼각에 대한 하나의 입구를 가진다.

파일에 배정된 매개 쪼각에 대한 하나의 입구를 가진다. 대표적으로 파일색인들은 파일배정표의 부분으로서 물리적으로 기억되지 않는다. 오히려 파일에 대한 파일색인은 개별적인 블로크에 유지되며 파일배정표에서 파일에 대한 입구는 그 블로크를 가리킨다. 배정은 블로크들(그림 12-11)이나 가변길이쪼각들(그림 12-12)에 기초할수 있다. 블로크들에 의한 배정은 외부분렬을 제거하지만 반면에 가변길이쪼각들에 의한 배정은 국소성을 증가시킨다. 어느 경우이든 파일통합을 자주 수행할수 있다. 파일통합은 가변길이쪼각들의 경우에 색인크기를 줄이지만 블로크배정인 경우에는 그렇지 못하다. 색인에 의한 배정은 순차접근과 직접접근을 둘다 지원하며 따라서 파일배정의 가장 일반적인 형식이다.

자유공간관리

파일이름	시작블로크	길이
• • •	• • •	• • •
파일 B	0	5
• • •	• • •	• • •

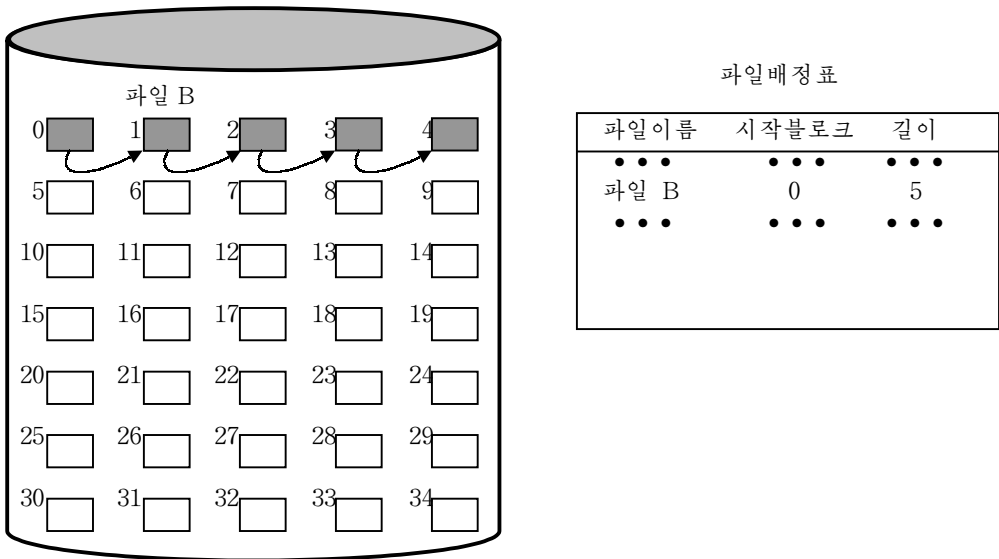



그림 12-10. 사슬배정 (통합정리후)

자유공간관리

파일에 배정된 공간을 관리하여야 하는것과 같이 현재 임의의 파일에 배정되지 않은 공간도 역시 관리하여야 한다. 앞에서 서술한 임의의 파일배정수법을 적용하기 위하여서는 디스크의 어느 블로크들을 사용할수 있는가를 알아야 한다. 따라서 파일배정표에 추가하여 **디스크배정표**가 필요하다. 여기서는 이미 실현된 여러가지 수법들을 론의한다.

비트표

이 방법은 디스크의 매개 블로크에 대하여 한개 비트를 포함하는 벡토르를 사용한다. 0 으로 된 매개 입구는 자유블로크에 대응하며 1 로 된 매개 입구는 사용중에 있는 블로크에 대응한다. 실례로 그림 12-7 의 디스크배치도에서 길이가 35 인 벡토르가 필요하며 그것은 다음의 값을 가질것이다.

00111000011111000011111111111011000

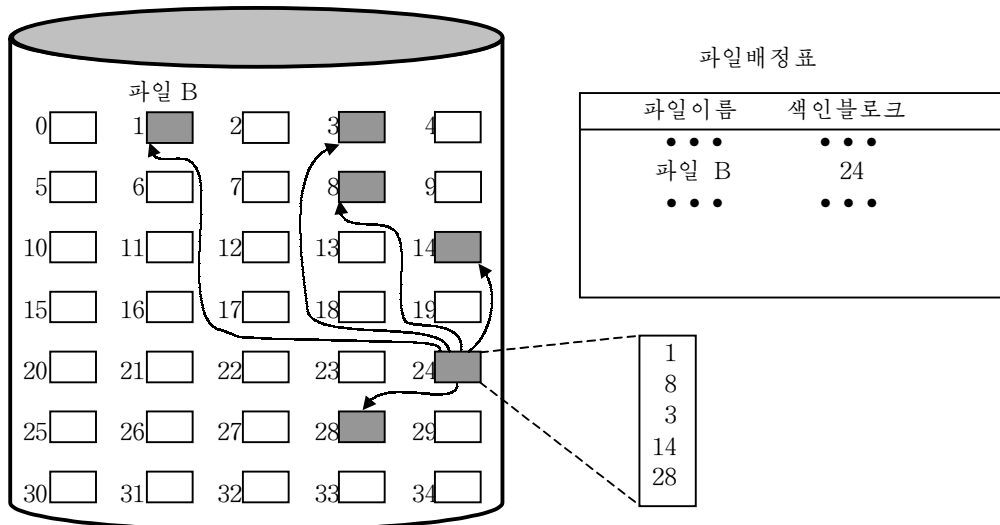



그림 12-11. 블로크쪼각을 가지는 색인에 의한 배정

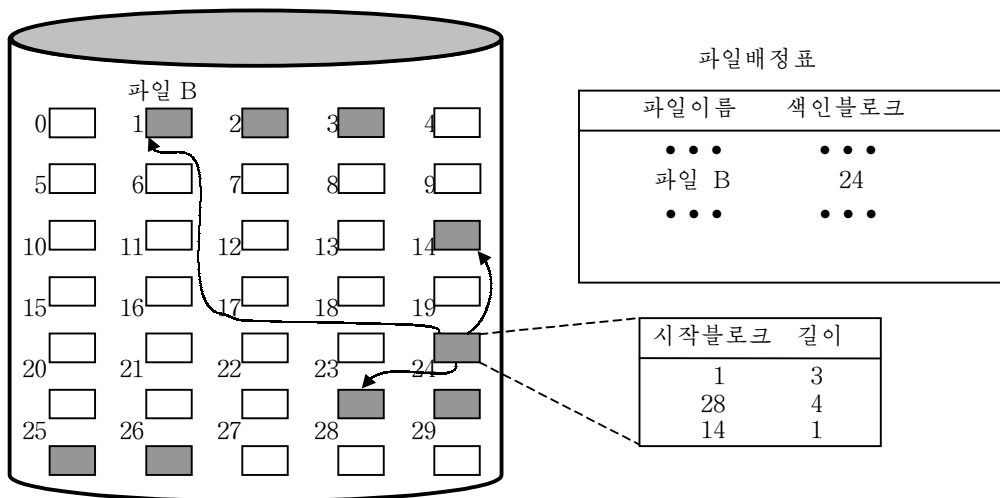



그림 12-12. 가변길이쪼각을 가지는 색인에 의한 배정

비트표는 한개 블로크 또는 린접하는 블로크들의 그룹을 찾기가 상대적으로 쉽다는 우점을 가진다. 따라서 비트표는 륜곽적으로 설명한 임의의 파일배정방법들에 매우 적합하다. 다른 우점은 비트표가 될수록 작다는것이다. 그러나 적당한 크기일수도 있다. 블로크비트배치표에 필요한 기억기용량(바이트)은 다음과 같다.

$$\frac{\text{바이트단위의 디스크크기}}{8 \times \text{파일체계의 블로크크기}}$$

따라서 512bit의 블로크들을 가지는 16Gbyte디스크에서 비트표는 약 4Mbyte를 차지한다. 비트표를 위하여 4Mbyte 의 주기억기를 내여 줄수 있는가? 만약 그렇다면 디스크접근을 요구하지 않고 비트표를 탐색할수 있다. 그러나 오늘날의 기억기크기의 경우조

차도 4Mbyte 는 단일기능에 사용하기에는 매우 큰 주기억기이다. 다른 방도는 디스크에 비트표를 배치하는것이다. 그러나 4Mbyte 의 비트표는 약 8000 개의 디스크블로크를 요구할것이다. 블로크가 요구될 때마다 그만한 크기의 디스크공간을 탐색할수 없으므로 기억기에 비트표를 상주시키고 있다.

비트표가 주기억기에 있을 때조차도 표의 완전한 탐색은 용납할수 없을 정도로 파일체계의 성능을 약화시킬수 있다. 이것은 특히 디스크가 거의 차고 자유블로크가 얼마 남지 않았을 때 그러하다. 따라서 비트표를 사용하는 대부분의 파일체계는 비트표의 국부령역들의 내용을 요약하는 보조자료구조들을 유지한다. 실례로 표는 많은 동일크기의 국부령역들로 론리적으로 분할될수 있다. 요약표는 매개 국부령역에 대한 자유블로크들의 수와 자유블로크들의 최대크기의 린접수를 포함한다. 파일체계는 많은 린접블로크들이 필요하면 요약표를 조사하여 적당한 국부령역을 찾은 다음 그 국부령역을 탐색할수 있다.

사슬식자유쪼각

매개 자유쪼각의 지시자와 길이값을 사용하여 자유쪼각들을 사슬로 련결할수 있다. 이 방법은 디스크배정표를 전혀 요구하지 않으며 이따금씩 사슬머리에로의 지시자와 첫 쪼각의 길이를 요구하기때문에 공간내부처리는 무시해도 좋다. 이 방법은 모든 파일배정방법에 적합하다. 한번에 한개 블로크를 배정한다면 사슬의 머리부에서 간단히 자유블로크를 선택하고 첫 지시자와 길이값을 조절한다. 가변길이쪼각을 배정한다면 첫블로크배정알고리듬을 사용할수 있다. 즉 사슬에서 다음번 적합한 자유쪼각을 결정하기 위하여 쪼각들에서 머리부들을 한꺼번에 꺼낸다. 역시 지시자와 길이값들을 조절한다.

이 방법은 자기고유의 문제를 가진다. 얼마간 사용한후에 디스크는 작은 쪼각들로 분렬되고 많은 쪼각들은 단일블로크길이를 가지게 된다. 역시 블로크를 배정할 때마다 그 블로크에 자료를 쓰기전에 먼저 블로크를 읽어 새로운 첫 자유블로크에로의 지시자를 되찾을 필요가 있다. 파일조작을 위하여 많은 개별적블로크들을 동시에 배정할 필요가 있다면 이 방법은 파일생성을 매우 느리게 한다. 이와 류사하게 사방으로 분렬되여 있는 파일들을 삭제하는 조작은 매우 많은 시간을 소비한다.

색인작성

색인작성방법은 자유공간을 파일로 취급하며 파일배정에서 서술된바와 같이 색인표들을 사용한다. 효률을 위하여 블로크가 아니라 가변크기쪼각들에 기초하고 있다. 따라서 디스크에 있는 매개 자유쪼각을 위하여 표에 한개 입구를 가진다. 이 방법은 모든 파일배정방법들을 효과적으로 지원한다.

자유블로크목록

이 방법에서는 매개 블로크에 순차적으로 번호가 할당되며 모든 자유블로크들의 번호목록이 디스크의 예약구역에 유지된다. 디스크의 크기에 따라서 24bit 또는 32bit 가 단일블로크번호를 기억하는데 요구된다. 그래서 자유블로크목록의 크기는 대응하는 비트표의 크기의 24 배 또는 32 배로 되며 따라서 주기억기가 아니라 디스크에 기억시켜야 한다. 그러나 이것은 아주 좋은 방법이다. 다음의 문제들을 고찰하자

1. 자유블로크목록에 사용된 디스크공간이 총 디스크공간의 1%아래이다. 32 bit 블로크번호를 사용하면 벌칙공간은 매 512byte 블로크당 4byte 이다.

2. 자유블로크목록이 너무 커서 주기억기에 기억시킬수 없어도 주기억기에 목록의 작은 부분을 기억시키기 위한 두가지 효과적인 수법이 있다.

ㄱ) 목록은 첫 수천개 요소가 주기억기에 있는 밀어넣기탄창(부록 1-ㄴ)으로 취급될수 있다. 새로운 블로크는 배정될 때 주기억기에 있는 탄창의 꼭대기에서 꺼내 진다. 이와 류사하게 블로크는 해방될 때 탄창에 채워 진다. 다만 기억기에 있는 탄창부분이 다 차거나 비게 되였을 때 디스크와 주기억기사이의 자료이송이 있어야 한다. 따라서 이 수법은 대부분 거의 령시간접근을 제공한다.

ㄴ) 목록은 주기억기에 있는 대기렬의 머리부와 꼬리부에 수천개의 입구를 가진 FIFO 대기렬로 취급될수 있다. 블로크는 대기렬의 머리부로부터 첫번째 입구를 취하는것으로 배정되며 대기렬의 꼬리부의 끝에 입구를 추가하는것으로 해제된다. 다만 주기억기에 있는 대기렬의 머리부분이 비게 되거나 주기억기에 있는 대기렬꼬리부분이 다 채워 지면 디스크와 주기억기사이에 자료이송이 있어야 한다.

앞에서 말한 두가지 전략중 어느 전략(탄창 또는 FIFO 대기렬)에서도 배경스레드는 린접배정을 쉽게 하기 위하여 주기억기내부목록 또는 목록들을 천천히 분류할수 있다.

신뢰성

다음의 씨나리오를 고찰하자.

1. 사용자 A 는 현존 파일에로 자료를 추가하기 위하여 파일배정을 요청한다.
2. 요청이 허락되여 주기억기의 파일배정표들이 갱신되지만 디스크의것은 갱신되지 않는다.
3. 체계는 폭주하여 결과 재시동한다.
4. 사용자 B 는 파일배정을 요청하며 사용자 A 의 마지막배정과 겹치는 디스크공간에 배정된다.
5. 사용자 A 는 A 의 파일내부에 기억된 참조를 통하여 겹쳐 진 쪼각에 접근한다.

이 난관은 체계가 효률을 위하여 주기억기에 디스크배정표와 파일배정표의 사본을 유지했기때문에 발생하였다. 이러한 형태의 오유를 막기 위하여 파일배정이 요청될 때 다음의 단계들을 수행할수 있다.

1. 디스크에 있는 디스크배정표를 페쇄한다. 이것은 이 배정이 끝날 때까지 표가 교체되는것을 막는다.
2. 사용가능한 공간을 위하여 디스크배정표를 탐색한다. 이것은 디스크배정표의 사본이 항상 주기억기에 유지되여 있다는것을 가정한다. 그렇지 않으면 디스크배정표를 주기억기에 읽어 내야 한다.
3. 공간을 배정하고 디스크배정표를 갱신하며 디스크를 갱신한다. 디스크의 갱신은 디스크에 있는 디스크배정표를 다시 작성할것을 요구한다. 사슬식디스크배정에서 디스크의 갱신도 역시 디스크에 있는 디스크배정표를 다시 작성할것을 요구한다.
4. 파일배정표를 갱신하고 디스크를 갱신한다.
5. 디스크배정표를 해제한다.

이 수법은 오유를 방지할수 있다. 그러나 작은 쪼각들이 자주 배정될 때 성능에 미치는 영향은 매우 클것이다. 이 간접소비를 줄이기 위하여 묶음배정기구를 사용할수 있다. 이 경우에 디스크의 자유쪼각들의 묶음은 배정을 위하여 얻어 진다. 디스크의 대응하는 쪼각들에는 《사용중》이라는 표식이 붙여 진다. 이 묶음을 사용하는 배정은 주기억기에서 수행할수 있다. 묶음이 다 사용되면 디스크의 디스크배정표가 갱신되며 새로운 묶음을 획득할수 있다. 체계폭주가 발생하면 《사용중》이라는 표식이 붙은 디스크의 부분들은 재배정되기전에 어떤 방법으로든지 표식이 지워 져야 한다. 삭제수법은 파일체계의 고유한 특징들에 관계된다.

제 7 절. UNIX 의 파일관리

UNIX 핵심부는 모든 파일들을 바이트들의 흐름으로 본다. 임의의 내부론리구조는 응용에 따라 독특하다. 그러나 UNIX 는 파일들의 물리적구조에 관계된다.

파일은 네가지 형태로 구분된다.

- **보통** : 사용자, 응용프로그람 또는 체계편의프로그람이 입구한 정보를 포함하는 파일들.
- **등록부** : 후에 서술하는 관련된 마디들(색인마디들)의 지시자들과 함께 파일이름으로 된 목록을 포함한다. 등록부들은 계층적으로 구성된다(그림 12-4). 등록부파일들은 실제상 사용자프로그람에게는 읽기접근이 가능하며 파일체계만이 쓰기접근을 할수 있도록 특수한 쓰기보호특권들을 가진 보통 파일들이다.
- **특수** : 말단과 인쇄기와 같은 주변장치에 접근하는데 사용된다. 매개 입출력장치들은 제 11 장 제 7 절에서 론의된바와 같이 특수파일과 관련된다.
- **이름붙임** : 제 6 장 제 7 절에서 론의된바와 같이 이름붙인 흐름관

이 절에서는 대부분의 체계들이 파일로서 취급되는 보통파일들의 처리에 대하여 보게 된다.

i 마디

조작체계는 i 마디들을 사용하여 모든 형태의 UNIX 파일들을 관리한다. i 마디(정보마디)는 특정한 파일에 대하여 조작체계가 요구하는 주요한 정보를 포함하는 자료구조이다. 여러가지 파일이름들은 단일 i 마디에 관련될수 있지만 능동 i 마디는 정확히 한개 i 마디로 조종된다.

파일의 허가 및 다른 조종정보들은 물론 파일의 속성들도 i 마디에 기억된다. 표 12-4 는 그 내용을 제시한다.

파일배정

파일배정은 블로크에 기초하여 수행된다. 배정은 미리배정을 사용하지 않고 필요에 따라 동적으로 진행된다. 그러므로 디스크에 있는 파일블로크들은 반드시 린접하지는 않는다. 색인방식은 매개 파일을 추적하기 위하여 사용되는데 파일을 위하여 i 마디에 기억된 색인부분을 가진다. i 마디는 13 개의 3byte 주소 또는 지시자들로 구성된 39byte 주소정보를 포함한다. 첫 10 개 주소는 파일의 첫 10 개 자료블로크를 가리킨다. 파일이 10 개 블로크보다 더 크면 한개 준위이상의 간접수단을 다음과 같이 사용한다.

표 12-4. 디스크상주의 UNIX 1 마디의 정보

파일방식	파일과 관련한 접근 및 집행의 허가를 기억하는 16bit 기발 12-14 파일형태(규칙,등록부, 문자형 또는 블로크형, FIFO 관) 9-11 집행기발들 8 소유자읽기허가 7 소유자쓰기허가 6 소유자집행허가 5 그룹읽기허가 4 그룹쓰기허가 3 그룹집행허가 2 다른 사용자읽기허가 1 다른 사용자쓰기허가 0 다른 사용자집행허가
련결계수	이 i 마디에로의 등록부참조의 수
소유자 ID	파일의 개별적소유자
그룹 ID	이 파일과 관련된 그룹소유자
파일크기	39byte 의 주소정보
마지막접근	마지막파일접근시간
마지막변경	마지막파일변경시간
i 마디변경	마지막 i 마디변경시간

- i 마디의 11 번째 주소는 색인의 다음쪼각을 포함하는 디스크의 블로크를 가리킨다. 이것을 단일간접블로크라고 한다. 이 블로크는 파일의 다음블로크의 지시자들을 포함한다.
- 파일이 더 많은 블로크들을 포함하면 i마디의 12번째 주소는 2중간접블로크를 가리킨다. 이 블로크는 추가적인 단일간접블로크들의 주소목록을 포함한다. 매개 단일간접블로크들은 파일블로크에로의 지시자들을 차례로 포함한다.
- 파일이 그보다 더 많은 블로크들을 포함한다면 i마디의 13번째 주소는 세번째 준위의 색인인 3중간접블로크를 가리킨다. 이 블로크는 보통 2중간접블로크들을 가리킨다.

이것들을 모두 그림 12-13에서 설명한다. 파일의 총 자료블로크수는 체계의 고정길이블로크의 용량에 관계된다. UNIX체계 V에서 블로크의 크기는 1Kbyte이며 매개 블로크는 총 256개의 블로크주소를 유지할수 있다. 따라서 이 기구의 최대파일크기는 16Gbyte이상이다(표 12-5).

이 기구는 여러가지 우점을 가진다.즉

1. i 마디는 고정크기이며 상대적으로 작기때문에 주기억기에 오래동안 유지될수 있다.
2. 파일이 작을수록 간접호출이 전혀 없거나 적으며 처리 및 디스크접근시간을 줄인다.
3. 파일의 리론적인 최대크기는 실제상 모든 응용들을 만족시킬수 있도록 충분히 크다.

표 12-5. UNIX 파일의 용량

준위	블로크수	바이트수
직접	10	10K
단일간접	256	256K
2 중간접	256 × 256=65K	65M
3 중간접	256 × 65K=16M	16G

제 8 절. WINDOWS 2000의 파일체계

Windows 2000(W2K)는 Windows 95, MS-DOS, OS/2에서 실행하는 파일배치표(FAT)와 함께 수많은 파일체계들을 지원한다. 그러나 W2K개발자들은 워크스테이션들과 봉사기들의 고준위말단요구들을 만족시키기 위하여 새로운 파일체계인 W2K파일체계(NTFS)를 설계하였다. 고준위말단응용들의 실례로 다음과 같은것들을 들수 있다.

- 파일봉사기들, 계산봉사기들, 자료기지봉사기들과 같은 의뢰기/봉사기응용들
- 자원집합공학 및 과학응용들
- 큰 기관체계들을 위한 망응용들

이 절에서는 NTFS를 개괄한다.

NTFS의 기본특징

NTFS는 앞으로 보게 되는바와 같이 품위 있고 간단한 파일체계모형우에 구축된 유연하고 강력한 파일체계이다. NTFS의 가장 주목할만한 특징들은 다음과 같다.

- **회복가능성**: 새로운 W2K파일체계에 대한 요구목록에서 제일 중요한것은 체계폭주 및 디스크고장으로부터 회복하는 능력이였다. 이러한 고장들이 발생하는 경우에 NTFS는 디스크기록권들을 재구축할수 있으며 그것들을 모순이 없는 상태로 되돌린다. NTFS는 이것을 파일체계의 변화들에 대한 트랜잭션처리모형을 사용하여 수행한다. 매개 중대한 변화는 완전히 수행되거나 전혀 수행되지 않는 자동작용으로 취급된다. 고장시에 처리중에 있던 매개 트랜잭션은 결과적으로 원점으로 돌려 지거나 완성된다. 게다가 NTFS는 림계파일체계자료를 위하여 여유기억기를 사용하기때문에 디스크분구의 고장으로 파일체계의 구조와 상태를 서술하는 자료를 잃지 않는다.
- **보안**: NTFS는 W2K객체모형을 사용하여 보안을 실시한다. 열린 파일은 자기의 보안속성들을 규정하는 보안서술자를 가진 파일객체로서 실현된다.
- **대형디스크와 대형파일**: NTFS는 FAT와 함께 대부분의 다른 파일체계들보다 더 효률적인 용량이 큰 디스크들과 매우 큰 파일들을 지원한다.
- **다중자료흐름**: 파일의 실제내용은 바이트들의 흐름으로 취급된다. NTFS에서는 단일파일에 대하여 다중자료흐름들을 정의할수 있다. 이러한 특징을 가진 편의프로그람의 실례는 원격 Macintosh체계들이 W2K를 사용하여 파일들을 기억시키

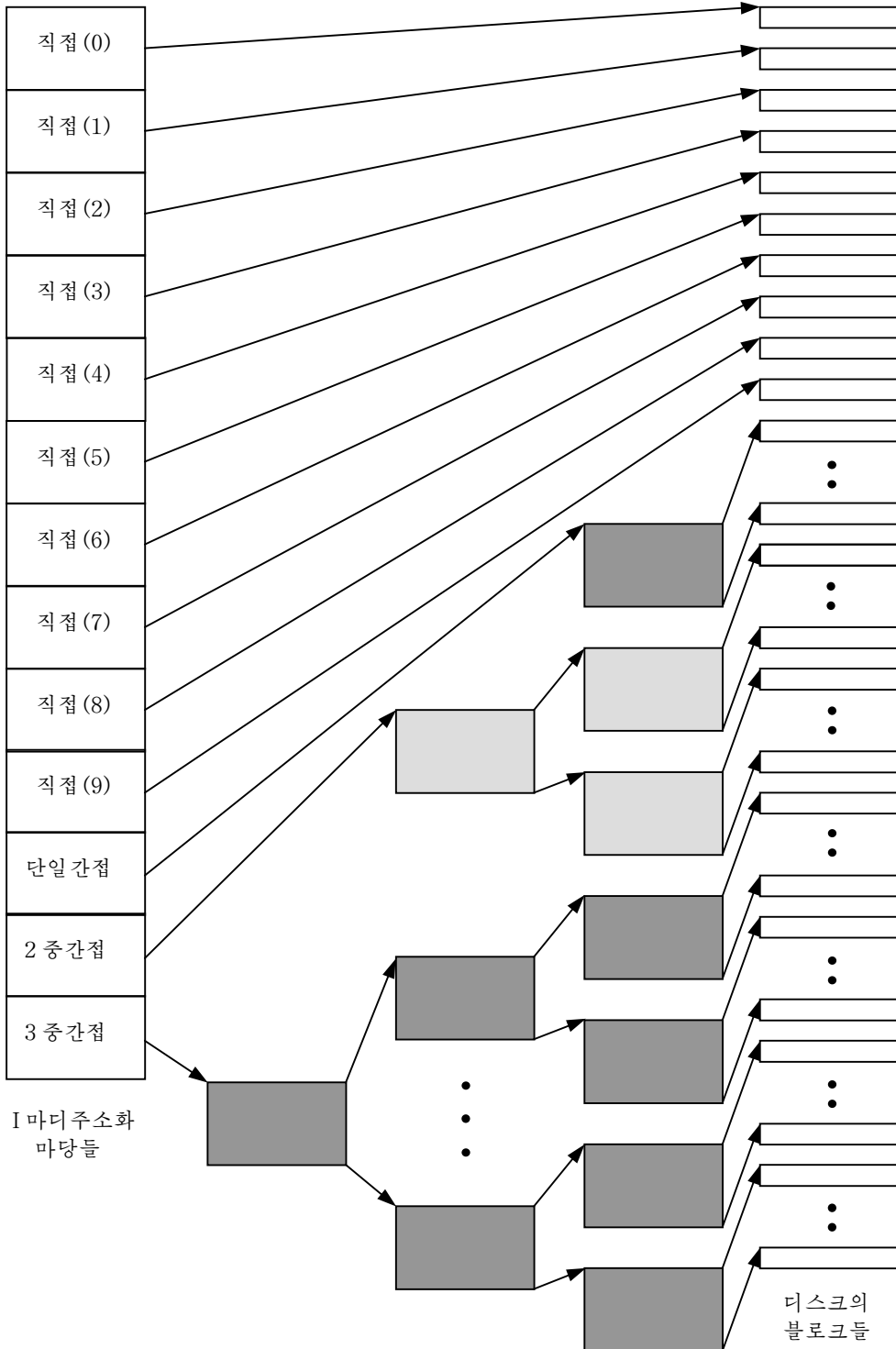



그림 12-13. UNIX 블로크주소지정방안

고 검색하도록 하는것이다. Macintosh에서 매개 파일은 두개 부분품 즉 파일자료와 파일에 대한 정보를 포함하는 자원갈래를 가진다. NTFS는 이 두개 부분품들을 두개의 자료흐름으로 취급한다.

- **일반색인작성기능**: NTFS는 속성들의 집합을 매개 파일과 관련시킨다. 파일관리체계에서 파일서술들의 모임은 관계자료기지로 구성되기때문에 임의의 속성으로 파일들의 색인을 작성할수 있다.

NTFS기록권과 파일구조

NTFS는 다음과 같은 디스크기억개념들을 사용한다.

- **분구**: 디스크의 가장 작은 물리적기억단위. 바이트단위의 자료크기는 2의 제곱이며 대체로 항상 512byte이다.
  - **클라스터**: 한개이상의 린접(같은 자리길에서 서로 린접)분구들. 분구단위의 클라스터크기는 2의 제곱이다.
- **기록권** : 한개이상의 클라스터들로 구성되며 공간배정을 위하여 파일체계가 사용하는 디스크의 론리적분할구역. 기록권은 언제나 파일체계정보와 파일들의 집합, 파일에 배정할수 있는 기록권의 임의의 보충적인 배정되지 않은 나머지 공간으로 구성된다. 기록권은 단일디스크 전체 또는 일부일수 있으며 또는 다중디스크들로 확장할수 있다. 하드웨어 또는 쏘프트웨어 RAID 5를 받아 들이면 기록권은 다중디스크들에 분산되여 구성된다. NTFS의 최대기록권크기는 2^{64}byte이다.

클라스터는 분구들을 인식하지 못하는 NTFS에서 배정의 기본단위이다. 실례로 매개 분구는 512byte이고 체계는 클라스터당 두개 분구(한 클라스터=1Kbyte)로 구성된다. 사용자가 1600byte용 파일을 생성하면 두개 클라스터가 파일에 배정된다. 사용자가 파일을 3200byte로 갱신하면 또다른 두개 클라스터가 배정된다. 파일에 배정된 클라스터들은 린접하지 않아도 된다. 따라서 디스크에 파일을 분렬시킬수 있다. 현재 NTFS가 지원하는 최대파일크기는 최대 2^{48}byte와 같은 2^{32}클라스터이다.

클라스터들을 배정에 사용하여 NTFS를 물리적분구크기와 독립시킨다. 이것으로 하여 NTFS는 512byte의 분구크기를 가지지 않는 비표준디스크들을 쉽게 지원할수 있다. 능률은 파일체계가 매개 파일에 배정하는 매개 클라스터를 추적하여야 한다는 사실로부터 높아 진다. 그리고 클라스터가 커짐에 따라 관리하여야 할 항목들도 적어진다.

표 12-6. Windows NTFS 분할구역과 클라스터크기

기록권크기	클라스터당 분구수	클라스터크기
≦ 512Mbyte	1	512bytes
512Mbyte-1Gbyte	2	1K
1Gbyte-2Gbyte	4	2K
2Gbyte-4Gbyte	8	4K
4Gbyte-8Gbyte	16	8K
8Gbyte-16Gbyte	32	16K
16Gbyte-32Gbyte	64	32K
＞ 32Gbyte	128	64K

표 12-6은 NTFS의 기정클라스터크기들을 보여 준다. 기정값은 기록권의 크기에 관계된다. 특정한 기록권에 사용하는 클라스터크기는 사용자가 기록권을 초기화(형식화)할것을 요청할 때 NTFS가 설정한다.

NTFS기록권배치도

NTFS는 디스크기록권에서 정보를 구성할 때 매우 단순하면서 강력한 방법을 사용한다. 기록권우의 매개 요소는 파일이며 매개 파일은 속성들의 집합으로 구성된다. 지어 파일의 자료내용들도 속성으로 취급된다. 이러한 간단한 구조인 경우에는 몇가지 범용기능들만으로도 파일체계를 구성하고 관리하는데 충분하다.

그림 12-14는 네개의 령역으로 구성된 NTFS기록권의 배치도를 보여 준다. 임의의 기록권의 첫 몇개 분구들은 부트시동정보와 코드는 물론 기록권배치도와 파일체계구조들에 대한 정보를 포함하는 **분할구역부트분구**(분구라고 하지만 16개 분구까지 될수 있다.)가 차지한다. 그다음에 배정되지 않은 사용가능한 공간에 대한 정보는 물론 이 NTFS기록권의 모든 파일들과 서류철들(등록부들)에 대한 정보를 포함하는 **주파일표**(MFT)가 있다. MFT는 본질상 이 NTFS기록권의 모든 내용들의 목록이며 관계자료기지구조에서 행들의 모임으로 구성된다.

MFT의 다음에는 **체계파일**들을 포함하는 대표적으로 약 1Mbyte크기의 령역이 있다. 이 령역의 파일들에는 다음과 같은것들이 있다.

- **MFT2** : 단일분구가 고장인 경우에 MFT에로의 접근을 보증하는데 사용되는 MFT의 첫 세개 행들의 대칭복제
- **운영일지파일** : NTFS회복가능성에 사용되는 트랜잭션단계들의 목록
- **클라스터비트사영**: 어느 클라스터가 사용중에 있는가를 보여 주는 기록권의 표현
- **속성정의표** : 이 기록권에서 지원되는 속성형태들을 정의하며 그것들을 색인으로 작성할수 있는가, 없는가 그리고 체계회복조작을 하는동안에 그것들을 회복할수 있는가, 없는가를 지적한다.

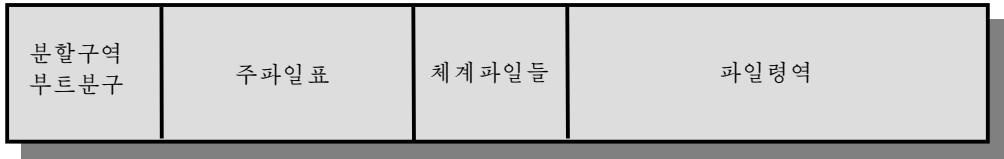



그림 12-14. NTFS기록권배치도

주파일표

W2K파일체계의 심장부는 MFT이다. MFT는 레코드라고 하는 가변길이행들의 표로 구성된다. 매개 행은 그자체가 파일로서 취급되는 MFT와 함께 이 기록권에 있는 파일 또는 서류철을 서술한다. 파일의 내용이 매우 작으면 파일전체는 MFT행에 배치된다.

그렇지 않은 경우에 그 파일을 위한 행은 부분적인 정보를 포함하며 파일의 나머지

는 기록권의 다른 사용가능한 클라스터들에 배치한다. 클라스터들의 지시자들은 그 파일의 MFT 행에 지적한다.

MFT 에서 매개 레코드는 파일(또는 서류철)특징들과 파일내용들을 정의하는데 필요한 속성들의 모임으로 구성된다. 표 12-7 은 행에서 찾아 볼수 있는 속성들을 명암법으로 지적하여 필요한 속성들과 함께 제시한다.

표 12-7. Windows NTFS 파일과 등록부속성형태

속성형태	서술
표준정보	접근속성들(읽기, 읽기/쓰기 기타 등등), 파일이 생성된 또는 마지막으로 변경된 시간 등의 시간압인, 얼마나 많은 등록부가 파일을 지적하는가(련결수) 등의 정보를 포함한다.
속성목록	파일과 매 속성이 위치하는 MFT 파일레코드의 파일참조를 작성하는 속성들의 목록. 모든 속성들이 단일 MFT 파일레코드에 맞지 않을 때 사용된다.
파일이름	파일 또는 등록부는 한개이상의 이름을 가져야 한다.
보안서술자	파일을 소유하고 파일에 접근하는 사용자를 규정한다
자료	파일의 내용. 파일은 하나의 이름이 없는 기정자료속성을 가지며 한개이상의 이름 있는 자료속성들을 가질수 있다.
색인뿌리	서류철을 실현하는데 사용된다.
색인배정	서류철을 실현하는데 사용된다.
기록권정보	기록권의 판본과 이름과 같은 기록권관련정보들을 포함한다.
비트사영	MFT 또는 서류철에 쓰이고 있는 레코드들을 표현하는 사영을 준다.

주의: 어두운 부분은 파일속성이 필요하다는것을 의미한다. 다른 속성은 선택적이다.

회복능력

NTFS는 체계폭주 또는 디스크고장이후에 파일체계를 모순이 없는 상태로 회복할수 있도록 한다. 회복능력을 지원하는 주요한 요소들은 다음과 같다(그림 12-15).

- **입출력관리자**: NTFS의 열기, 닫기, 읽기, 쓰기 등 기본기능들을 조정하는 NTFS 구동프로그람을 포함한다. 그밖에 쏘프트웨어 RAID모듈 FTDISK를 구성하여 사용할수 있다.
- **운영일지파일봉사** : 디스크쓰기에 대한 등록을 유지한다. 운영일지파일은 체계고장인 경우에 NTFS형식기록권을 회복하는데 사용된다.
- **캐쉬관리자**: 성능을 개선하기 위하여 파일읽기와 쓰기를 고속완충하는 프로그람. 캐쉬관리자는 제11장 제8절에서 서술한 차후쓰기와 차후완료수법들을 사용하여 디스크입출력을 최량화한다.
- **가상기억기관리자** : NTFS는 파일참조들을 가상기억기참조들로 사영하고 가상기억기를 읽고 쓰는것으로 고속완충된 파일들에 접근한다.

NTFS가 사용하는 회복수속들은 파일내용이 아니라 파일자료를 회복하기 위하여 설계되였다. 따라서 사용자는 폭주때문에 기록권 또는 응용의 등록부/파일구조를 결코 잃지 않을것이다. 그러나 파일체계는 사용자자료를 보증하지 않는다. 사용자자료를 포함하여 충분한 회복능력을 보장하자면 더욱더 복잡하며 자원을 소비하는 회복기구가 필요할

것이다.

NTFS 회복능력의 본질은 등록이다. 파일체계를 변경하는 매개 조작은 트랜잭션으로 취급된다. 중요파일체계자료구조들을 변경하는 트랜잭션의 매개 보조조작은 디스크기록권에 등록되기전에 운영일지파일에 등록된다. 등록을 사용하면 폭주시에 부분적으로 완성된 트랜잭션은 후에 다시 수행될수 있거나 체계가 회복되였을 때 원상태로 돌아 올수 있다.

[CUSTP4]에서 서술된바와 같이 일반적으로 회복능력을 보증하기 위하여 몇가지 단계들이 필요하다.

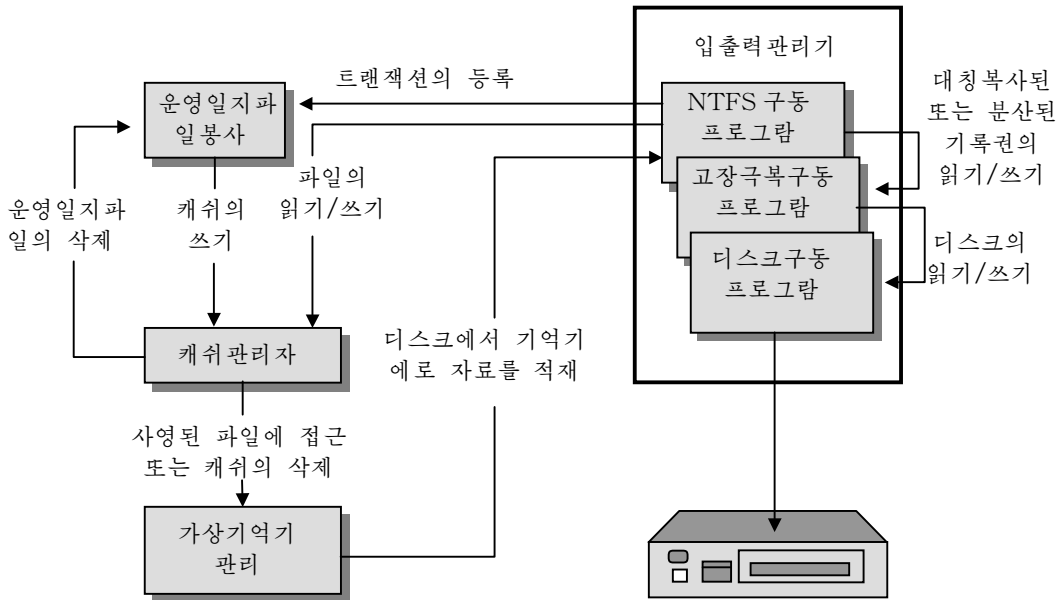



그림 12-15. WindowsNTFS 부분품들[CUST94].

1. NTFS 는 운영일지파일체계를 호출하여 기록권구조를 변경시킬 임의의 트랜잭션들을 캐쉬에 있는 운영일지파일에 등록한다.
2. NTFS 는 (캐쉬에 있는) 기록권을 변경시킨다.
3. 캐쉬관리자는 운영일지파일체계를 호출하여 그것이 운영일지파일을 디스크에 써 넣도록 한다.
4. 운영일지파일이 디스크에 안전하게 갱신되면 캐쉬관리자는 기록권변화들을 디스크에 써넣는다.

요약, 기본용어 및 복습문제

파일관리체계는 파일접근, 등록부보수, 접근조종을 비롯하여 사용자와 응용들에서 파일들을 사용하기 위한 여러가지 봉사를 제공하는 체계쏘프트웨어의 모임이다. 대표적으로 파일관리체계는 그자체가 조작체계의 한 부분이 아니라 조작체계가 봉사하는 체계봉사로 볼수 있다. 그러나 임의의 체계에서 파일관리기능의 일부는 어쨌든 조작체계로 수행된다.

파일은 레코드들의 집합으로 조직된다. 이 레코드들에 접근하는 방법은 디스크에 있

는 파일의 론리적조직과 어느정도까지는 파일의 물리적조직을 결정한다. 파일이 주로 전체로서 처리된다면 순차조직이 가장 간단하고 가장 적합하다. 순차접근이 필요하지만 개별적인 파일에로의 임의의 접근도 요구된다면 이때에는 색인달린 순차파일이 가장 좋은 성능을 줄수 있다. 파일접근이 주로 임의로 진행된다면 색인에 의한 파일 또는 하쉬파일이 가장 적합할것이다.

어떤 파일구조를 선택하든지 등록부봉사는 역시 필요하다. 이것은 파일들을 계층방식으로 구성하도록 한다. 이러한 구성은 사용자가 파일들을 추적하는데 편리하며 파일관리체계가 사용자에게 접근조종과 다른 봉사들을 제공하는데 쓸모 있다.

일반적으로 파일레코드는 고정길이로 구성되였을 때조차도 물리적인 디스크블로크의 크기와 같아 지지 않는다. 따라서 몇가지 부류의 블로크작성전략이 필요하다. 복잡성, 성능, 공간사용들사이의 관계는 사용되는 블로크작성전략을 결정한다.

임의의 파일관리기구의 기본기능은 디스크공간의 관리이다. 이 기능의 한 부분이 파일에 디스크블로크들을 배정하기 위한 전략이다. 여러가지 방법들이 도입되였으며 매개 파일의 배정을 추적하는데 여러가지 자료구조가 사용되였다. 더우기 배정되여 있지 않는 디스크공간을 관리하여야 한다. 이 후자의 기능은 주로 어느 블로크들이 사용가능한가를 표시하는 디스크배정표를 유지하는것이다.

기본용어

접근방식	파일이름	색인달린 순차파일	마당
비트표	파일배정표	색인마디	파일
블로크	파일등록부	열쇠마당	색인에 의한 파일배정
사슬식파일배정	파일관리체계	경로명	
린접파일배정	파일이름	더미	작업등록부
자료기지	하쉬파일	레코드	
디스크배정표	색인달린 파일	순차파일	

복습문제

1. 마당과 레코드의 차이는 무엇인가 ?
2. 파일과 자료기지의 차이는 무엇인가 ?
3. 파일관리체계란 무엇인가 ?
4. 파일조직을 선택하는데서 중요한 기준은 무엇인가 ?
5. 다섯가지 파일조직방법을 간단히 정의하시오.
6. 파일의 레코드를 찾기 위한 평균탐색시간이 순차파일에서 보다 색인달린 순차파일에서 더 작은 리유는 무엇인가 ?
7. 등록부에서 수행할수 있는 대표적인 조작들은 무엇인가 ?
8. 경로명과 작업등록부사이의 관계는 무엇인가 ?
9. 특정파일에 대하여 특정사용자에게 부여되거나 거절되는 대표적인 접근권들은 무엇인가 ?
10. 세가지 블로크작성방법들을 서술하고 간단히 정의하시오.
11. 세가지 파일배정방법들을 서술하고 간단히 정의하시오.

참 고 문 헌

파일관리에 대한 훌륭한 책은 많다. 다음의 책들은 모두 파일관리체계들을 집중적으로 취급할뿐아니라 관련조작체계문제들도 힘을 넣어 설명한다. 파일관리에 대한 정량적방법들을 취급하며 디스크일정작성으로부터 파일구조에 이르기까지 그림 12-2 에서 제시하는 모든 문제들을 취급하는 [WIED87]이 아마 가장 유용할것이다. [LIVA90]은 파일구조들을 강조하여 설명하며 비교성능분석에 대한 훌륭하고 구체적인 고찰을 진행한다. [GROS86]은 파일입출력방법들과 파일접근방법들에 다 관련되는 균형탐색문제들을 취급한다. 그것은 또한 파일체계에 필요한 모든 조종구조들에 대하여 일반적으로 서술한다. 이것들은 파일체계설계를 평가하는데서 쓸모 있는 검사목록을 제공한다. [FOLK98]은 파일들의 처리문제들과 유지, 탐색, 분류, 공유와 같은 문제들을 강조하여 설명한다. [CUST94]는 NT 파일체계에 대하여 훌륭히 개괄하며 [NAGA97]은 이 문제들을 더 상세히 고찰한다.

CUST94 Custer, H. *Inside the Windows NT File System*. Redmond, WA: Microsoft Press, 1994.

FOLK98 Folk, M., and Zoellick, B. *File Structures: An Oriented Approach with* C++. Reading, MA: Addison-Wesley, 1998.

GROS86 Grosshans, D. *File Systems:Design and Implementation*. Englewood Cliffs, NJ: Prentice Hall, 1986.

LIVA90 Livadas, P. *File Structures: Theory and Practice*. Englewood Cliffs, NJ: Prentice Hall, 1990.

NAGA97 Nagar, R. *Windows NT File System Internals,* Sebastopol, CA: O'Reilly, 1997.

WIED87 Wiederhold, G. *File Organization for Database Design.* New York: McGraw Hill, 1987.

련 습 문 제

1. 정의

   B = 블로크의 크기
   R = 레코드의 크기
   P = 블로크지시자의 크기
   F = 블로크작성결수; 블로크안에 기대되는 레코드의 수

   그림 12-6 에서 설명한 세가지 블로크작성방법에 대하여 F에 대한 식을 작성하시오.

2. 미리배정의 문제점과 린접쪼각의 랑비 또는 부족문제를 회피하는 한가지 방법은 파일이 성장하는데 따라 크기가 증가하는 쪼각들을 배정하는것이다. 실례로 한개 블로크크기의 쪼각으로부터 시작하여 매개 배정마다 쪼각의 크기를 배로 늘

인다. 블로크작성결수 F를 가진 n개 레코드로 된 파일을 고찰하자. 그리고 파일배정표로는 간단한 1준위색인을 사용한다고 가정하자

ㄱ) 파일배정표에서 입구수의 웃한계를 F와 n의 함수로 제시하시오.
ㄴ) 임의의 시각에 배정될 사용되지 않은 파일공간의 최대크기는 얼마인가 ?

3. 자료가 다음의 상태일 때 접근속도, 기억공간의 사용, 갱신(추가, 삭제, 변경)에 대한 용이성의 측면에서 효률을 최대로 하기 위하여 어떤 파일조직을 선택해야 하는가 ?

ㄱ) 자료는 드문히 갱신되고 임의의 순서로 자주 접근된다.
ㄴ) 자료는 자주 갱신되고 상대적으로 전체로서 접근된다.
ㄷ) 자료는 자주 갱신되고 임의의 순서로 자주 접근된다.

4. 등록부는 제한된 방법으로만 접근할수 있는 《특수파일들》로서 또는 보통자료파일로서 실현될수 있다. 매개 방법의 우점과 결함들은 무엇인가 ?

5. 일부 조작체계들은 나무구조파일을 가지지만 나무의 깊이는 작은 수의 준위로 제한된다. 이 제한은 사용자들에게 어떤 영향을 미치는가? 이것은 파일체계설계를 어떻게 간단화하는가(파일체계설계를 한다면)?

6. 자유공간목록에 자유디스크공간을 유지하는 계층파일체계를 고찰하자.

ㄱ) 자유공간에로의 지시자를 잃어 버렸다고 가정하자. 체계는 자유공간목록을 재구축할수 있는가?

ㄴ) 지시자는 단일기억기고장으로 결코 잃어 지지 않는다는것을 담보하는 기구를 제기하시오.

7. UNIX 파일의 조직이 i마디(그림 12-13)로 표시되였다고 하자.

매개 i마디에 12개의 직접블로크지시자와 단일, 2중, 3중간접지시자가 있다고 가정하자. 그리고 체계블로크크기와 디스크분구크기는 둘다 8K라고 가정하자. 디스크블로크지시자는 32bit이고 그중 8bit는 물리적디스크를 식별하고 24bit는 물리적블로크를 식별한다면

ㄱ) 이 체계가 지원하는 최대파일크기는 얼마인가 ?

ㄴ) 이 체계가 지원하는 최대파일체계분할구역은 얼마인가 ?

ㄷ) 파일의 i마디가 이미 주기억기에 존재하는것외에 다른 정보는 전혀 없다고 가정한다면 위치 13423956에 있는 바이트에 접근하는데 몇번의 디스크접근이 필요한가?

제 6 편. 분산체계

제 6 편의 중심

자료처리기능은 전통적으로 집중방식으로 구성되였다. 집중자료처리방식에서 자료처리는 중앙자료처리기지에 있는 일반적으로 대형콤퓨터들로 구성된 클라스터 또는 한대의 대형콤퓨터로 지원한다. 이러한 기지에서 수행되는 많은 과제들은 중앙에서 생성된 결과와 함께 중앙에서 개시된다. 완전한 집중자료처리기지는 많은 의미에서 집중되여 있다.

- **집중형콤퓨터 :** 중앙기지에는 한대 또는 그이상의 콤퓨터가 있다. 이 대형콤퓨터들은 대체로 공기조화기와 훌륭한 마루와 같은 특수설비들을 요구한다. 보다 작은 조직에서 중앙콤퓨터 또는 콤퓨터들은 큰 미니콤퓨터들 또는 중간규모의 체계들이다. IBM의 AS/400계렬들은 중간규모체계의 실례이다.
- **집중처리 :** 모든 응용프로그람들은 중앙자료처리기지에서 실행한다. 이 기지는 특정한 기관단위로 사용자들의 요구를 지원하는 응용프로그람은 물론 생활비지불과 같은 사실상 명백히 중앙 또는 기관규모의 응용프로그람들을 포함한다. 후자의 실례로서 제품설계부서는 중앙기지에서 실행하는 콤퓨터지원설계(CAD)도형제품을 사용할수 있다.
- **집중된 자료 :** 모든 자료는 중앙기지에 있는 파일과 자료기지에 보관되며 중앙콤퓨터또는 콤퓨터들로 조종되고 접근된다. 이것은 오직 한개 기관단위의 요구들을 지원하는 자료는 물론 재고목록수자들과 같이 많은 기관단위들에 쓸모 있는 자료를 포함하며 많은 기관단위들에 의하여 사용될것이다. 후자의 실례로 시장기관은 구매자조사들로부터 얻은 정보로 자료기지를 유지할수 있다.

이렇게 집중된 기관은 많은 매력적인 전망들을 가진다. 장비와 쏘프트웨어의 구입과 운영단계를 줄일수 있다. 큰 중앙 DP상점은 여러 부서들의 요구들을 만족시키기 위하여 전문적인 프로그람작성자들을 종업원으로 채용할수 있다. 관리부는 자료처리획득을 조종하고 프로그람작성과 자료파일구조에 대한 규격들을 실시하며 보안방책을 설계하고 실현한다.

자료처리기지는 분산자료처리(DDP)전략을 실현함으로써 여러 측면에서 집중자료처리기관과 다를수 있다. 분산자료처리기지는 콤퓨터들 보통은 소형콤퓨터들이 기관의 여기저기에 분산되여 있는 기지이다. 이러한 분산의 목적은 운영, 경제 그리고 지정학적고려에 기초하여 더 효과적인 방법으로 정보를 처리하기 위해서이다. DDP기지는 중앙기지와 함께 위성기지들을 포함할수 있으며 또는 동등한 계산기지들의 공동체와 거의 류사할수 있다. 어느 경우이든 일반적으로 호상 련결형식이 필요하다. 즉 체계의 여러가지 콤퓨터들이 서로 련결되여야 한다. 기대되는바와 같이 여기서 규정된 집중자료처리의 특징이 결정되면 DDP기지는 콤퓨터들과 처리, 자료의 분산을 요구한다.

DDP의 우점은 다음과 같다.

- **응답성 :** 국부계산기지들은 중앙기지에 있으면서 전체 기관의 요구들을 만족시키는 z 기지보다 국부기관의 경영관리요구들을 더 잘 직접적으로 만족시킬수 있게 관리될수 있다.
- **사용성 :** 다중상호련결체계들의 경우에 어느 한 체계의 상실은 최소한의 영향을 미칠것이다. 기본체계들과 부분품들(실례로 정밀한 응용프로그람들과 인쇄기들,

대량기억장치들)은 여벌체계가 고장난후에 부하를 재빨리 넘겨 받을수 있도록 복제될수 있다.

- **자원공유** : 비싼 하드웨어는 사용자들사이에 공유될수 있다. 자료파일들은 기관규모의 접근권을 가지고 집중적으로 관리되고 유지될수 있다. 부원봉사들과 프로그람들 그리고 자료기지들은 기관규모들로 개발될수 있으며 분산기지들에 분산될수 있다.
- **점차적인 성장** : 집중기지에서 증가된 작업부하 또는 새로운 응용프로그람들에 대한 요구는 보통 주요한 장비구입 또는 주요한 쏘프트웨어갱신을 필요로 한다. 이것은 중요한 지출을 동반한다. 게다가 주요한 변화는 오유위험과 성능저하를 초래하는 현존응용들의 변환 또는 재작성을 요구할수 있다. 분산체계에서는 응용프로그람들 또는 체계들을 점차적으로 교체하여《전부 교체 또는 령교체》방법을 회피할수 있다. 더우기 응용프로그람을 새로운 기계로 옮기는데 필요한 비용이 허용되지 않는다면 낡은 장비를 기지에 남겨 두고 단일응용프로그람을 실행하게 할수 있다.
- **사용자관련의 증가와 조종** : 보다 소형이면서 관리하기 쉬운 장비가 사용자에게 물리적으로 가까이 배치됨에 따라 사용자는 기술성원들과의 지휘대화에 의해서든가 자기의 직속상관을 통하여 체계설계의 운영에 영향을 줄수 있는 더 많은 기회를 가지게 된다.
- **말단사용자생산성** : 분산체계는 사용자에게 보다 빠른 응답시간을 보장한다. 그것은 매개 장비부분들이 보다 작은 일감들을 처리하려고 하기때문이다. 또한 기지들의 응용프로그람들과 대면부들은 기관단위의 요구들에 최대한으로 활용될수 있다. 단위관리자들은기지의 국부부분의 효과를 평가하고 적당하게 변경하는 지위에 있다.

이러한 리익들을 달성하기 위하여 조작체계는 DDP 가 지원하는 기능들의 범위를 규정하여야 한다. 여기에는 기계들사이의 자료교환을 위한 쏘프트웨어, 높은 리용성과 성능을 달성하기 위하여 기계들을 클라스터화하는 능력, 분산환경에서 프로쎄스들을 관리할수 있는 능력들이 포함된다.

제 6 편의 안내

제 13 장. 분산처리, 의뢰기/봉사기 및 클라스터

제 13 장은 협동하여 동작하는 다중체계들에 필요한 조작체계지원을 고찰한다. 이 장에서는 더욱더 중요한 개념으로 되고 있는 의뢰기/봉사기의 계산방식과 이 방식이 조작체계에 제기하는 요구들을 고찰한다. 의뢰기/봉사기계산방식에 대한 론의는 의뢰기봉사기체계들의 실현에 사용되는 두가지 기본도구들 즉 통보문넘기기와 원격수속호출들에 대한 설명을 동반한다. 제 13 장은 또한 클라스터의 개념을 설명한다.

제 14 장. 분산형프로쎄스의 관리

제 14 장은 분산조작체계를 개발하는데서 제기되는 기본문제들을 연구한다. 먼저 프로쎄스이주에 대한 요구들과 기구들을 분석한다. 프로쎄스이주로 능동프로쎄스는 부하평형 또는 사용성에 대한 목적을 달성하기 위하여 자기의 생존기간에 한 기계에서 다른 기계에로 이동한다. 다음으로 분산조작체계의 개발에서 사활적인 요소인 분산전역상태의 개념을 고찰한다. 마지막으로 분산환경에서 호상배제와 교착과 관련된 병행성을 연구한다.

제 13 장. 분산처리, 의뢰기/봉사기 및 클라스터

값이 눅으면서 성능이 좋은 개인용콤퓨터들과 봉사기들의 사용률이 높아 짐에 따라 처리기들과 자료 그리고 자료처리체계의 다른 측면들을 기관안에서 분산시킬수 있는 분산자료처리(DDP)에로 나가는 경향이 높아 지고 있다. DDP 체계는 계산기능의 분할을 포함하며 또한 자료기지와 장치조종, 대화(망)조종의 분산조직을 포함한다.

많은 기관들에서는 봉사기와 결합된 개인용콤퓨터를 매우 중요시하고 있다. 개인용콤퓨터는 문서처리기, 표계산프로그람, 도형표현과 같은 사용자에게 친숙한 여러가지 응용프로그람들을 지원하는데 사용된다. 봉사기는 복잡한 자료기지관리기능을 가진 기관의 자료기지와 정보체계쏘프트웨어를 보관한다. 개인용콤퓨터들사이에 그리고 매개 개인용콤퓨터와 봉사기사이에 결합이 필요하다. 개인용콤퓨터를 간이말단으로 취급하는것으로부터 개인용콤퓨터응용프로그람들과 봉사기자료기지사이를 높은 수준에서 통합하는것에 이르기까지의 범위에서 여러가지 방법들이 일반적으로 사용되고 있다.

이러한 응용추세는 조작체계와 지원프로그람들에서 분산성능의 발전에 의하여 안받침되였다. 성능범위가 조사되였다.

- **통신구성방식 :** 독립적인 콤퓨터들로 이루어 진 망을 지원하는 쏘프트웨어이다. 그것은 전자우편, 파일전송, 원격말단접근과 같은 분산응용들에 필요한 지원을 준다. 그러나 콤퓨터는 사용자와 응용들에 대하여 서로 다른 신원을 유지한다. 사용자와 응용들은 명백한 참조로 다른 콤퓨터들과 통신하여야 한다. 매개 콤퓨터가 자기의 독자적인 조작체계를 가지며 모든 콤퓨터들이 같은 통신구성방식을 지원하는한 서로 다른 종류의 콤퓨터와 조작체계를 혼합할수 있다. 가장 널리 사용되고 있으며 가장 유명한 통신구성방식의 실례가 부록 1에서 서술된 TCP/IP 규약이다.
- **망조작체계 :** 응용기계들 보통 단일사용자워크스테이션들과 한대 또는 그이상의 《봉사기》기계들로 구성된 망이 있는 구성이다. 봉사기기계들은 파일기억과 인쇄기관리와 같은 망준위의 봉사와 응용프로그람을 보장한다. 매개 콤퓨터는 자기의 전용조작체계를 가진다. 망조작체계는 응용기계들이 봉사기기계들과 대화하도록 하는 국부조작체계의 부속물이다. 사용자는 여러개의 독립적인 콤퓨터가 존재한다는것을 알고 그것들을 정확히 다루어야 한다. 대표적으로 범용통신구성방식이 망응용프로그람들을 지원하는데 사용된다.
- **분산조작체계 :** 콤퓨터망이 공유하는 범용조작체계. 이것은 사용자들에게는 보통 중앙집중조작체계처럼 보이지만 다수의 기계자원들에 대한 투명접근을 사용자에게 제공한다. 분산조작체계는 기초통신기능들을 위한 통신구성방식에 관계된다. 보다 일반적으로는 불필요한 기능을 제거한 통신기능들을 조작체계에 통합하여 효률을 높인다.

모든 제작자들이 통신구성방식수법을 개발하고 지원하고 있다. 망조작체계는 보다 더 최근에 나타났지만 많은 제품들이 나왔다. 분산체계에 대한 연구개발은 분산조작체계령역에서부터 먼저 시작되였다. 몇가지 제품들이 출현하였지만 충분한 기능을 가진 분산조작체계는 아직 실험단계에 있다.

이 장과 다음 장에서는 분산처리성능을 고찰한다. 먼저 의뢰기/봉사기방식, 통보문넘기기와 원격수속호출을 비롯한 분산쏘프트웨어의 몇가지 기본 개념들에 대하여 고찰한다. 다음 더욱더 중요한 클라스터방식을 고찰한다.

제 14 장은 분산조작체계의 몇가지 주요한 견해를 고찰한다.

제 1 절. 의뢰기/봉사기의 계산작업

최근년간에 정보체계에서의 가장 중대한 경향은 아마 의뢰기/봉사기계산방식일것이다. 이 계산방식은 대형콤퓨터가 지배하는 집중계산방법들과 분산자료처리의 다른 방식들 모두를 급속히 교체하고 있다.

이 절에서는 먼저 의뢰기/봉사기계산방식의 일반특징을 서술한다. 다음으로 파일봉사기를 사용하는것으로 하여 제기되는 파일개시일관성문제를 조사한다. 마지막으로 미들웨어의 개념을 소개한다.

의뢰기/봉사기계산방식이란 무엇인가?

콤퓨터분야에서의 다른 새로운 부문들과 마찬가지로 의뢰기/봉사기계산방식은 자기의 통용어들을 가지고 있다. 의뢰기/봉사기제품들과 응용프로그람들에 대한 서술에서 일반적으로 찾아 볼수 있는 몇가지 용어들을 표 13-1에 주었다.

표 13-1. 의뢰기/봉사기용어

용어	설명
응용프로그람작성대면부(API)	의뢰기와 봉사기들이 호상 통신하도록 하는 함수들과 호출프로그람들의 모임
의뢰기	봉사기의 자료기지나 다른 정보들에 질문할수 있는 망정보요구자, 보통 PC 또는 워크스테이션
미들웨어	의뢰기응용프로그람과 봉사기사이의 련결성을 개선하는 구동프로그람들, API 들 또는 다른 응용프로그람들의 모임
관계자료기지	모든 탐색기준들을 만족시키는 행들을 선택하도록 정보접근을 제한하는 자료기지
봉사기	망의 의뢰기들이 조작하는 정보를 저장하는 고성능워크스테이션, 소형콤퓨터 또는 대형콤퓨터
구조화질문언어(SQL)	관계자료기지를 주소지정, 생성, 갱신, 질문하기 위하여 IBM 이 개발하고 ANSI 가 표준화한 언어

그림 13-1은 의뢰기/봉사기개념의 본질을 파악하도록 한다. 용어가 암시하는것과 같이 의뢰기/봉사기환경에서는 의뢰기들과 봉사기들이 동작하고 있다. 의뢰기들은 일반적으로 말단사용자에게 매우 친근한 사용자대면부를 주는 단일사용자 PC 들과 워크스테이션들이다. 의뢰기들은 일반적으로 창문들과 마우스를 사용하여 사용자들에게 가장 편리한 도형형태의 대면부를 준다. 이와 같은 대면부들의 공통실례로서 Microsoft 의 Windows 와 Macintosh 를 들수 있다. 의뢰기용응용프로그람들은 사용하기 쉽게 만들어지며 표계산프로그람과 같은 편리한 도구들을 포함한다.

의뢰기/봉사기환경에서 매개 봉사기는 의뢰기들에 사용자공유봉사모임을 제공한다. 현재 가장 일반적인 형태의 봉사기는 보통 관계자료기지를 조종하는 자료기지봉사기이다. 봉사기에 의하여 많은 의뢰기들은 같은 자료기지들을 공유하여 접근할수 있으며 고성능콤퓨터체계를 사용하여 자료기지를 관리할수 있다.

의뢰기와 봉사기외에 의뢰기/봉사기환경의 세번째 본질적인 구성요소는 망이다. 의뢰기/봉사기의 계산방식은 분산계산방식이다. 사용자들과 응용프로그람들 그리고 자원들은 업무요구에 따라 분산되며 단일 LAN 또는 WAN 또는 인터네트에 의하여 련결된다.

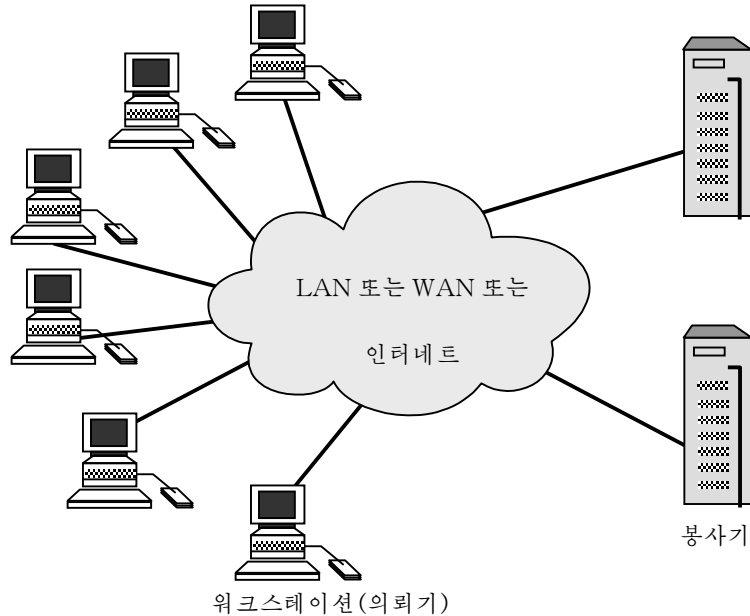



그림 13-1. 일반적인 의뢰기/봉사기환경

의뢰기/봉사기구성이 다른 분산처리방법과 어떻게 다른가? 의뢰기/봉사기를 강조하며 일반분산처리와 구별하는 특징은 다음과 같다.

- 사용자의 체계에 매우 신뢰성 있게 편리한 응용프로그람들을 보장한다. 이것은 사용자에게 콤퓨터사용형태와 동기에 대한 많은 조종권을 주며 부서수준의 관리자들에게 그들의 국부적요구에 대처할수 있는 능력을 준다.
- 응용프로그람들이 분산되더라도 기관의 자료기지들과 망관리 및 편의프로그람기능들을 집중시킨다는 우점이 있다. 이것으로 하여 기관의 경영자는 계산 및 정보체계에서 전반적인 총 투자액을 조종할수 있으며 체계들을 서로 련결할수 있도록 상호사용가능성을 제공한다. 동시에 콤퓨터에 기초한 복잡한 설비들을 유지하는데 필요한 많은 개별적인 부서들과 과들을 줄이며 자료 및 정보접근에 필요한 그 어떤 형태의 기계 및 대면부도 선택할수 있다.
- 사용자조직과 제작자들에 의하여 열린 모듈체계를 실현한다. 이것은 여러 제작자들의 제품을 선택하고 장비를 혼합하는데서 보다 자유롭다는것을 의미한다.
- 망화에서는 운영이 기본이다. 따라서 망관리와 망보호는 정보체계를 조직하고 운영하는데서 높은 우선권을 가진다.

의뢰기/봉사기응용

의뢰기/봉사기방식의 기본특징은 의뢰기와 봉사기사이에 응용프로그람수준의 과제들을 할당하는것이다. 그림 13-2는 일반 실례를 설명한다. 물론 의뢰기와 봉사기량쪽에서 기본쏘프트웨어는 하드웨어가동환경에서 실행되는 조작체계이다. 의뢰기와 봉사기의 가동환경들과 조작체계들은 서로 다를수 있다. 여러 개의 서로 다른 형태의 의뢰기가동환경과 조작체계들 그리고 여러 개의 서로 다른 형태의 봉사기가동환경과 조작체계들이 단

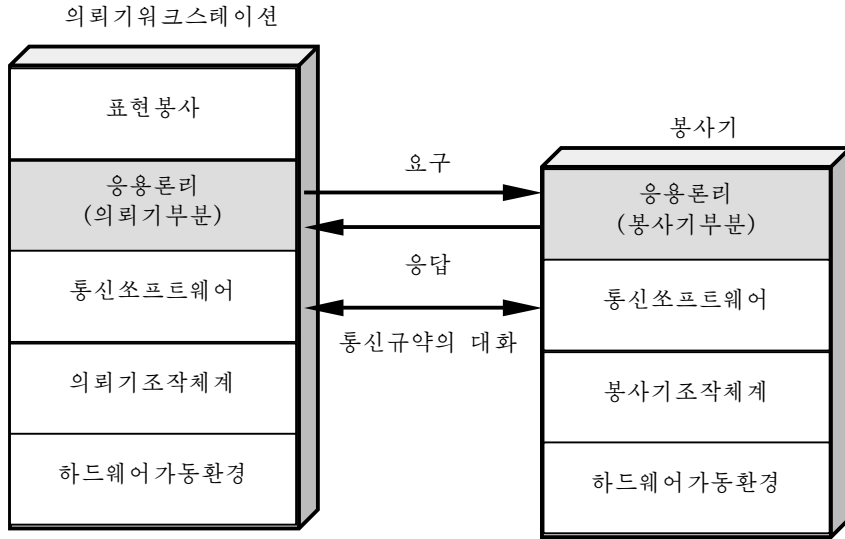



그림 13-2. 범용의뢰기/봉사기구성방식

일환경에 있을수 있다. 특정한 의뢰기와 봉사기가 같은 통신규약들을 공유하고 같은 응용프로그람들을 지원하는한 이보다 낮은 수준의 차이는 무의미하다.

통신쏘프트웨어는 의뢰기와 봉사기가 대화하도록 한다. 이러한 쏘프트웨어의 주요 실례가 TCP/IP이다. 물론 이 모든 지원쏘프트웨어(통신 및 조작체계)의 목적은 분산응용을 위한 기초를 주는것이다. 리상적으로는 응용프로그람이 수행하는 실제기능들을 가동환경과 망자원을 최대한으로 활용하도록 그리고 여러가지 과제를 수행하며 공유자원을 사용하는데서 다른 사용자와 협동하기 위하여 사용자들의 능력을 최대한으로 활용하도록 의뢰기와 봉사기에 나눌수 있다. 몇가지 실례에서 이러한 요구들은 큰 응용쏘프트웨어를 봉사기에서 집행할것을 규정한다. 이와는 반대로 다른 실례들에서는 대부분의 응용론리를 의뢰기에 배정한다.

의뢰기/봉사기환경을 성공시키는 본질적인 요인은 사용자가 체계와 전체로서 대화하는 방법에 있다. 따라서 의뢰기기계에 대한 사용자대면부의 설계는 매우 중요하다. 대부분의 의뢰기/봉사기체계에서는 사용하기 쉽고 배우기 쉬우면서도 강력하고 유연한 도형사용자대면부(GUI)를 보장하는것이 매우 중요하다. 따라서 의뢰기/봉사기환경에서 사용할수 있는 분산응용에 대한 매우 편리한 대면부를 줄수 있는 의뢰기워크스테이션에서의 표현봉사모듈을 생각할수 있다.

자료기지응용

의뢰기와 봉사기에 응용론리를 나누는 개념을 설명하는 실례로서 관계자료기지를 사용하는 가장 일반적인 의뢰기/봉사기응용프로그람들중의 하나를 고찰하자. 이 환경에서 봉사기는 본질적으로 자료기지봉사기이다. 의뢰기와 봉사기의 대화는 의뢰기가 자료기지를 요구하고 자료기지응답을 수신하는 트랜잭션형식으로 진행한다.

그림 13-3은 이러한 체계의 구성방식을 일반적인 용어로 설명한다. 봉사기는 자료기지를 유지해야 하며 이 목적을 위하여서는 복잡한 자료기지관리체계쏘프트웨어모듈이 요구된다. 자료기지를 사용하는 여러가지 서로 다른 응용프로그람들을 의뢰기기계들에 적

재할수 있다. 의뢰기와 봉사기를 이어 주는 《접착제》는 봉사기의 자료기지에 접근하기 위하여 의뢰기가 요청들을 발생하게 하는 쏘프트웨어이다. 이러한 론리의 일반적인 실례가
구조화질문언어(SQL)이다.

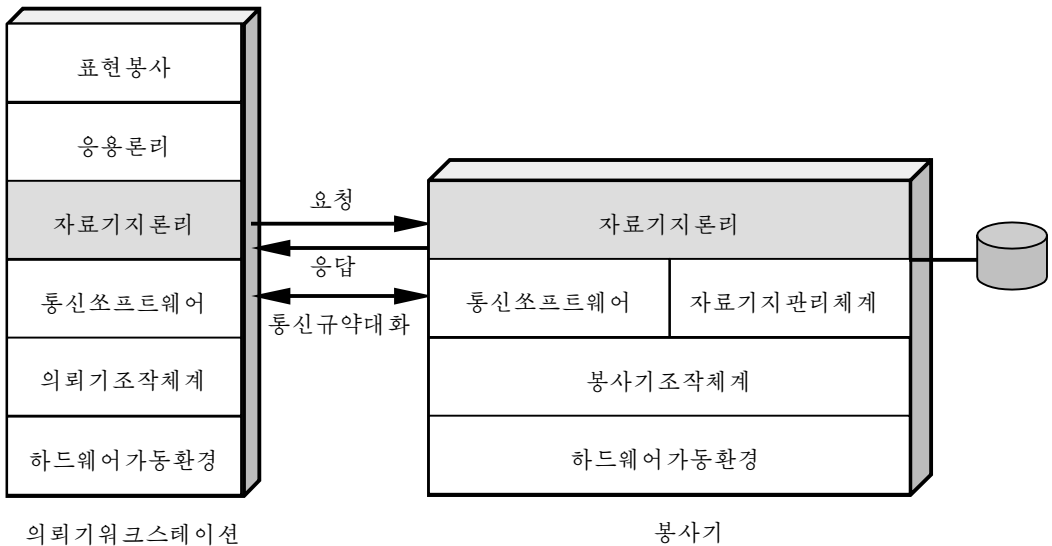



그림 13-3. 자료기지응용을 위한 의뢰기/봉사기구성방식

그림 13-3은 모든 응용론리 즉 복잡한 계산이나 다른 형태의 자료해석을 위한 쏘프트웨어는 의뢰기측에 있고 봉사기는 다만 자료기지관리에 관계한다고 가정한다. 이러한 구성이 적당한가 하는것은 응용프로그람의 형태와 목적에 따르다. 실례로 1차목적이 레코드조사를 위한 직결접근을 주는것이라고 가정하자. 그림 13-4 ㄱ는 이것이 어떻게 작업하는가를 보여 주고 있다. 봉사기는 백만레코드(관계자료기지용어에서는 행이라고 한다.)를 보관하고 있으며 사용자는 0, 한개 또는 기껏해서 몇개의 레코드를 산출하는 조사를 하려고 한다고 가정하자. 사용자는 여러 개의 탐색기준(실례로 1992년이전의 레코드, 오하이오주에 사는 사람들과 관계되는 레코드, 특수사건이나 특징들과 관계되는 레코드)을 사용하여 이 레코드들을 탐색할수 있을것이다. 맨 처음의 의뢰기질문은 탐색기준을 만족시키는 100,000개 레코드가 있다는 응답을 내보낼수 있다. 그다음에 사용자는 보충적인 제한조건을 추가하여 새로운 질문을 내보낸다. 이때 1000개의 가능한 레코드가 있다는것을 가리키는 응답이 되돌려 진다. 마지막으로 의뢰기는 보충제한조건을 가진 세번째 요청을 내보낸다. 마지막 탐색기준이 하나의 적합한 레코드를 산출하여 의뢰기에 다시 넘긴다.

앞에서 본 응용은 두가지 리유로 하여 의뢰기/봉사기구성방식에 잘 맞는다.

1. 자료기지를 분류하고 탐색하는 일감은 부피가 매우 크다. 이것은 큰 디스크나 디스크더미, 고속콤퓨터, 고속입출력구성방식을 요구한다. 이러한 용량과 능력은 단일사용자워크스테이션이나 PC에 필요 없으며 가격이 너무 비싸다.
2. 탐색을 위하여 의뢰기에 백만개 레코드파일전체를 이동시키면 망에 매우 큰 통화부담을 준다. 그러므로 봉사기가 직접 의뢰기를 대신하여 레코드들을 검색할수 있

는것만으로는 불충분하며 봉사기가 의뢰기를 대신하여 탐색을 수행할수 있게 하는 자료기지론리를 가져야 한다.

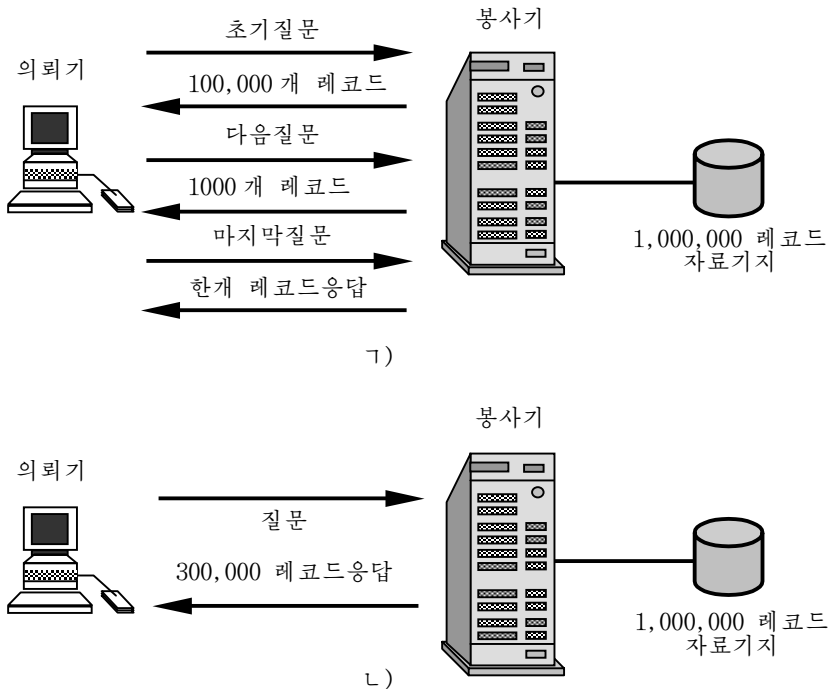



그림 13-4. 의뢰기/봉사기자료기지사용법
ㄱ-가능한 의뢰기/봉사기사용법, ㄴ- 오용된 의뢰기/봉사기

같은 백만개 레코드로 된 자료기지를 가지는 그림 13-4 ㄴ의 방안을 고찰하자. 이 경우에 한개 질문이 망을 통하여 300,000개 레코드를 전송한다. 이런 현상은 실례로 사용자가 많은 레코드들 또는 전체 자료기지에서 어떤 마당의 총 합계나 평균값을 구하려고 할 때 일어 난다.

명백히 이 후자의 방안은 받아 들이기 힘들다. 이 문제를 해결하기 위한 한가지 방법 즉 의뢰기/봉사기구성방식의 우점을 최대로 살리기 위한 방법은 봉사기에 응용론리의 일부를 넘겨 주는것이다. 즉 자료검색과 자료탐색은 물론 자료분석을 수행하기 위한 응용론리를 봉사기에 장비할수 있다.

의뢰기/봉사기응용의 종류

의뢰기/봉사기의 일반적인 틀거리내에서 의뢰기와 봉사기마다 다르게 일감을 나누어 여러가지 구성을 실현한다. 그림 13-5는 여러가지 처리배정방법을 설명한다. 그림은 자료기지를 응용하기 위한 몇가지 기본방법들을 개괄적으로 설명한다. 그밖의 방법들도 있는데 모든 방법들은 다른 형태의 응용을 위한 다른 특성서술을 가질수 있다. 어떤 경우나 이 그림을 보고 가능한 방법을 선택하는것이 좋다.

그림은 4가지 종류를 설명한다.

- **주콤퓨터에 의한 처리 :** 주콤퓨터에 의한 처리는 일반적으로 쓰는 용어로서 의뢰

기/봉사기계산방식이 아니라 오히려 실제상 모든 처리를 중앙의 주콤퓨터에서 수행하는 전통적대형콤퓨터환경이라고 할수 있다. 흔히 사용자대면부는 쏘기말단이다. 사용자가 개인용콤퓨터를 사용하고 있다고 해도 사용자국은 일반적으로 말단

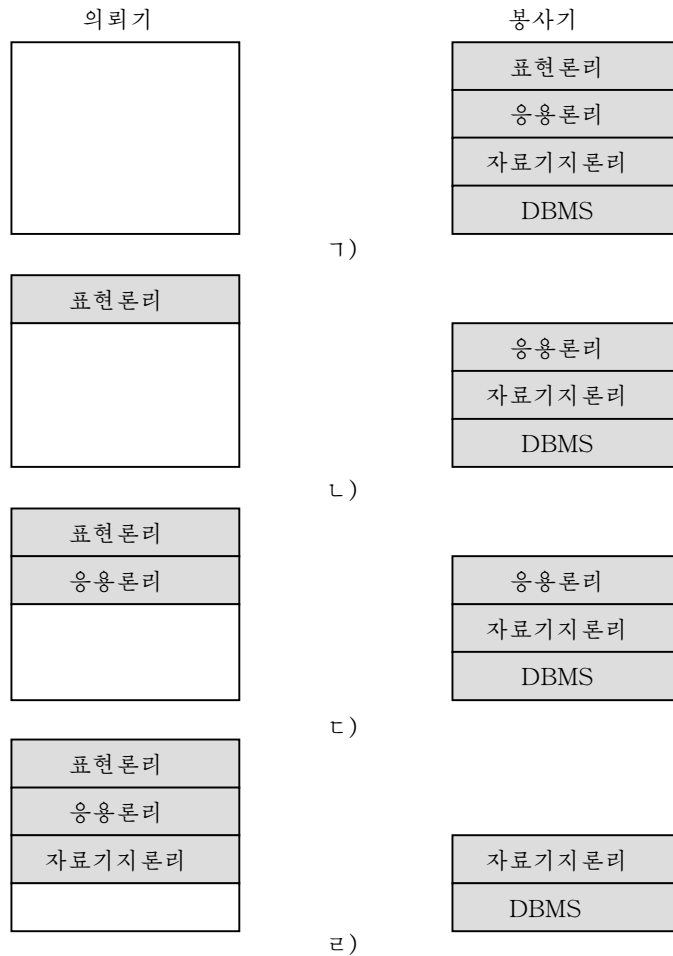



그림 13-5. 의뢰기/봉사기응용의 종류
ㄱ- 주콤퓨터에 의한 처리, ㄴ- 봉사기에 의한 처리, ㄷ- 협동처리, ㄹ- 의뢰기에 의한 처리

모의기의 역할만을 수행한다.

- **봉사기에 의한 처리:** 실제상 모든 처리는 봉사기에서 수행되고 의뢰기는 선차적으로 도형사용자대면부를 보장하는 가장 기본적인 의뢰기/봉사기구성이다. 이 구성은 초기의 의뢰기/봉사기체계 특히 부서수준의 체계를 대표한다. 이러한 구성이 합리적인것으로 되는것은 사용자워크스테이션이 사용자에게 편리한 대면부를 주는데 가장 적합하며 자료기지와 응용프로그람들을 중앙체계에 쉽게 유지할수 있기때문이다. 이러한 형태의 구성은 사용자가 더 좋은 대면부를 가진다는 우점이 있지만 생산성에서 큰 리득을 얻거나 체계를 지원하는 실제업무기능들을 근본적으로 개혁하는데는 적합하지 않다.
- **의뢰기에 의한 처리:** 다른 측면에서 대부분의 봉사기가 수행하는 자료확인루틴과 그밖의 자료기지론리기능들을 제외하고 실제상 모든 응용처리는 의뢰기에서 수

행할수 있다. 일반적으로 일부 좀 복잡한 자료기지론리기능들은 의뢰기에서 동작한다. 이 구성방식은 아마 현재 사용중에 있는 가장 일반적인 의뢰기/봉사기방법일것이다. 이 구성방식으로 하여 사용자는 국부적요구에 맞게 작성된 응용프로그람들을 사용할수 있다.

- **협동처리:** 협동처리구성에서는 응용처리를 최대로 활용할수 있는 방식으로 수행하며 의뢰기와 봉사기 두 기계의 우점과 자료분산의 우점을 가진다. 이러한 구성은 설정과 유지가 좀 복잡하지만 결국은 이러한 형태의 구성이 다른 의뢰기/봉사기방법들보다 사용자에게 더 큰 생산성리득과 더 큰 망효과성을 줄수 있다.

그림 13-5 ㄷ, ㄹ은 의뢰기에 적지 않은 부하를 주는 구성에 맞는다. 이른바 살찐 의뢰기모형은 Powersoft 회사의 PowerBuilder 와 Gupta 회사의 SQL Windows 와 같은 응용프로그람개발도구에 의하여 보급되여 왔다. 이 도구로 개발한 응용프로그람들은 대표적으로 부서범위에서 25~150 명의 사용자를 지원한다. 살찐 의뢰기모형의 기본리익은 그것이 탁상형콤퓨터능력의 우점을 가지며 봉사기에서 응용처리부담을 덜고 봉사기를 더 효률 있게 하며 병목이 적게 한다.

그러나 살찐 의뢰기전략은 몇가지 결함을 가진다. 의뢰기에 더 많은 기능을 추가함으로써 탁상기계들의 능력에 비하여 과중한 부하가 순식간에 걸리게 되므로 회사들로 하여금 장비를 개량할것을 요구한다. 이 모형이 부서범위를 벗어 나 확장되여 많은 사용자들이 협력하면 회사는 능력이 큰 LAN 들을 설치하여 여윈 봉사기들과 살찐 의뢰기들사이의 대량전송을 지원하여야 한다. 결국 수십, 수천대의 탁상형콤퓨터들에 분산시킨 응용프로그람들을 유지하고 갱신하고 교체하는것은 어렵다.

그림 13-5 ㄴ은 여윈 의뢰기방법을 보여 준다. 이 방법은 전통적인 주콤퓨터중심의 방법을 거의 모방한것이며 대형콤퓨터로부터 분산환경에로 기관규모의 응용들을 발전시키기 위한 이동행로이다.

3 층의뢰기/봉사기구성방식

전통적인 의뢰기/봉사기구성방식은 두개 수준(또는 층) 즉 의뢰기층과 봉사기층을 포함한다. 최근년간에는 3 층구성방식이 점차 일반화되였다(그림 13-6). 이 구성방식에서 응용쏘프트웨어는 세가지 형태의 기계 즉 사용자기계와 중간층봉사기, 후미봉사기에 분산된다. 사용자기계는 지금까지 론의해 온 의뢰기기계이며 3 층모형에서는 대표적으로 여윈 의뢰기이다. 중간층기계들은 본질적으로 여윈 사용자의뢰기들과 여러가지 후미자료기지봉사기들사이에 있는 망문이다. 중간층기계들은 통신규약들을 변환할수 있으며 한가지 형태의 자료기지질문을 다른 형태의 자료기지질문으로 넘길수 있다. 게다가 중간층기계는 서로 다른 원천들에서 발생한 결과들을 통합할수 있다. 결국 중간층기계는 탁상응용프로그람들과 후미의 유산응용프로그람들사이에서 두 세계를 중계하므로 관문으로서 봉사할수 있다.

중간층봉사기와 후미봉사기의 대화는 역시 의뢰기/봉사기모형에 따른다. 따라서 중간층체계는 의뢰기로서도 봉사기로서도 동작한다.

파일캐쉬일관성

파일봉사기를 사용할 때 망에 의하여 생긴 지연때문에 파일입출력성능은 국부파일접근에 비하여 현저하게 떨어 진다. 이 성능악화를 완화하기 위하여 개별적인 체계들은 최근에 접근한 파일레코드들을 보관하는 파일캐쉬를 사용할수 있다. 국부파일캐쉬를 사용하면 국소성의 원리로부터 수행하여야 할 원격봉사기에로의 접근회수를 줄일수 있다.

그림 13-7은 망에 집합된 워크스테이션들사이에서 파일들을 고속완충하기 위한 대표적인 분산기구를 설명한다. 프로쎄스가 파일에 접근할 때 먼저 프로쎄스워크스테이션의 캐쉬에 요구를 제출한다(《파일자료흐름》). 이 요청이 만족되지 않으면 파일이 국부디스

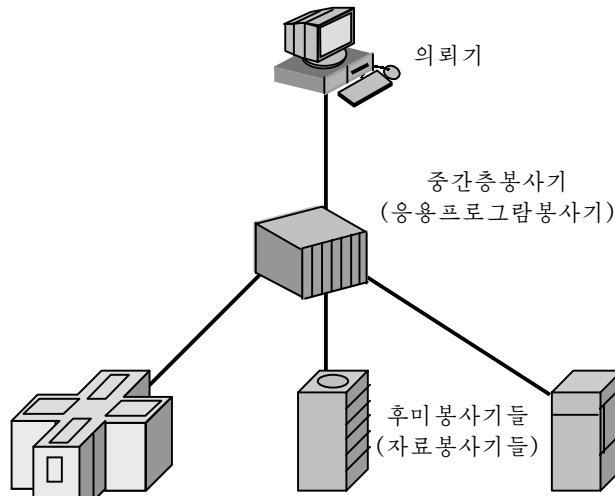



그림 3-6. 3층의뢰기/봉사기구성방식

크에 있는 경우에 요청을 국부디스크에 넘기거나(《디스크자료흐름》) 파일이 보관되여 있는 파일봉사기에 넘긴다(《봉사기자료흐름》). 봉사기에서는 먼저 봉사기의 캐쉬를 조사하고 없으면 봉사기의 디스크에 접근한다. 통신통화량(의뢰기캐쉬)과 디스크입출력(봉사기캐쉬)을 줄이기 위하여 2중고속완충방법을 사용한다.

캐쉬들이 항상 원격자료의 정확한 사본을 가지고 있으면 캐쉬들은 **일관성**이 있다고 말할수 있다. 원격자료가 변하고 그에 대응하는 낡은 국부캐쉬사본이 제거되지 않으면 캐쉬들은 일관성이 없을수 있다. 이런 현상은 한 의뢰기가 역시 다른 의뢰기에 의하여 고속완충된 파일을 변경하는 경우에 일어 날수 있다. 실제로 두가지 수준에서 난관이 존재한다.

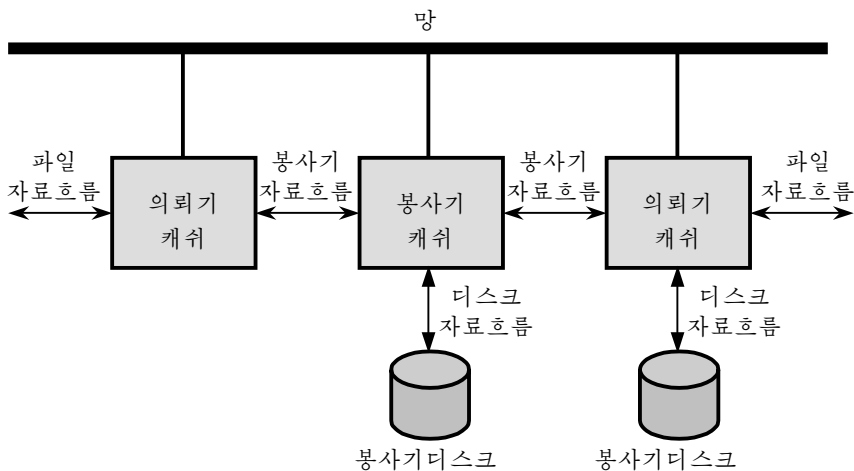



그림 13-7. Sprite 에서 분산파일고속완충

의뢰기가 봉사기에 파일의 임의의 변화를 즉시에 써넣는 방책을 받아 들이면 파일의 관련부분을 자기 캐쉬에 복사하고 있는 임의의 다른 의뢰기는 낡은 자료를 가지고 있게 될것이다. 의뢰기가 변화값을 봉사기에 좀 늦게 써넣으면 문제는 더 악화된다. 이 경우에 봉사기는 낡은 파일판본을 가지게 되며 봉사기에 새로운 파일읽기를 요구하면 낡은 자료를 얻을수 있다. 원격자료의 현재까지의 변화를 국부캐쉬복사로 유지하는 문제는 **캐쉬일관성**문제로서 알려 져 있다.

캐쉬일관성을 위한 가장 간단한 방법은 파일잠금수법을 사용하여 한대이상의 의뢰기가 한개 파일을 동시에 접근하는것을 막는것이다. 이것은 성능과 유연성을 희생시켜 일관성을 보증한다. Sprite의 기능들은 더 강력한 방법을 준다[NELS88, OUST88]. 임의의 수의 프로쎄스들은 읽기와 생성을 위하여 파일을 자기의 의뢰기캐쉬에 열수 있다. 그러나 봉사기에 대한 열린파일요청이 쓰기접근을 요구할 때와 그밖의 프로쎄스들이 읽기접근을 위하여 파일을 열 때 봉사기는 두가지 작용을 한다. 첫째로, 봉사기는 캐쉬를 가지고 있다고 해도 모든 변경된 블로크들을 즉시에 다시 써넣어야 한다는것을 쓰기프로쎄스에 통지한다. 기껏해서 이러한 의뢰기가 한대 있을수 있다. 둘째로, 봉사기는 파일을 여는 모든 읽기프로쎄스들에 파일을 더는 고속완충할수 없다는것을 알린다.

미들웨어

의뢰기/봉사기제품을 개발하고 전개하는 사업은 물리층으로부터 응용층까지 분산계산방식의 모든 견해들을 표준화하는 사업보다 훨씬 앞서 나가고 있다. 이 표준화의 결핍으로 통합되고 다중전문가를 지원하는 기업소규모의 의뢰기/봉사기구성을 실현하기가 힘들다. 의뢰기/봉사기방법의 리익은 업무의 해결방법을 주기 위하여 가동환경들과 응용프로그람들을 조합하고 맞물리게 하는 능력과 모듈성에 많이 관련되기때문에 정보처리상호운영문제를 해결하여야 한다.

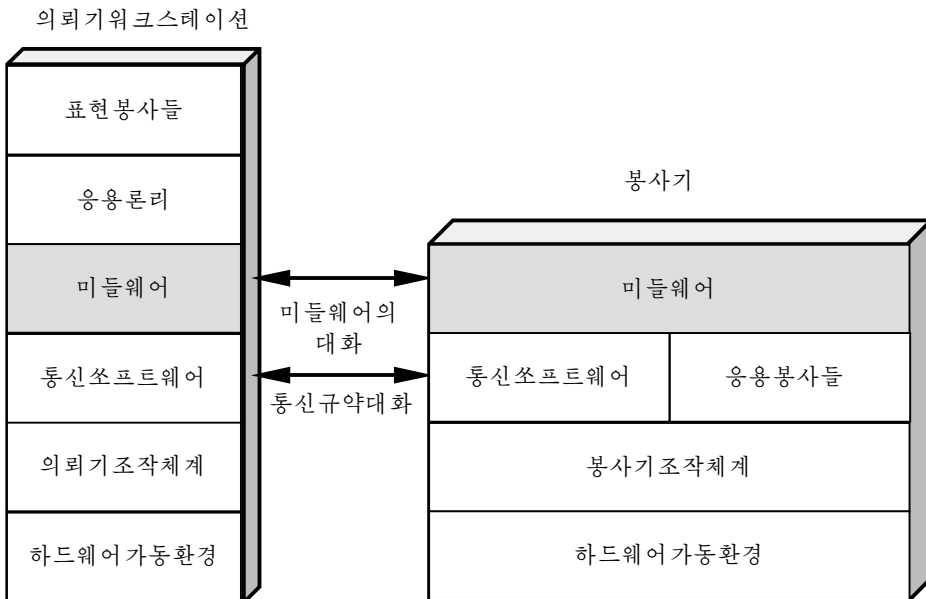



그림 13-8. 의뢰기/봉사기구성방식에서 미들웨어의 역할

의뢰기/봉사기방법의 근본리익을 달성하기 위하여 개발자들은 모든 가동환경들에 있는 체계자원들에로의 통일적인 접근방법과 접근형태를 주는 도구묶음을 가지고 있어야한다. 이것으로 하여 프로그람작성자는 여러가지 PC들과 워크스테이션들을 같은것으로 보고 느낄뿐아니라 같은 방법을 사용하여 자료의 위치에 관계 없이 자료에 접근하는 응용프로그람들을 작성할수 있다.

이 요구를 만족시키는 가장 일반적인 방법은 우에 있는 응용프로그람과 아래에 있는 통신쏘프트웨어 및 조작체계사이에 있는 표준프로그람작성대면부와 통신규약들을 사용하는것이다. 결국 이러한 표준화된 대면부와 통신규약들을 미들웨어라고 말하게 되였다. 표준프로그람작성대면부로는 여러가지 봉사기형태와 워크스테이션형태들에 대하여 같은 응용프로그람을 실현하기가 쉽다. 이것은 명백히 구매자에게는 리익으로 되며 제작자에게는 자극을 주어 이러한 대면부들을 보장하도록 한다. 그 리유는 구매자들이 봉사기가 아니라 응용프로그람들을 사기때문이다. 즉 구매자들은 자기들이 원하는 응용프로그람들을 실행하는 봉사기제품들만 선택할것이다. 이 여러가지 봉사기대면부들을 자기들에게로 접근하려는 의뢰기들에 련결하기 위하여 표준화된 규약들이 필요하다.

매우 단순한것으로부터 매우 복잡한것에 이르는 여러가지 미들웨어제품들이 있다. 그것들모두가 공통으로 가지고 있는것은 서로 다른 망규약들과 조작체계들의 복잡성과 차이들을 숨기는 능력이다. 의뢰기와 봉사기제작자들은 일반적으로 많은 보편화된 미들웨어제품들을 제공한다. 따라서 사용자는 특수한 미들웨어전략을 결정할수 있으며 그 전략을 지원하는 여러 제작자들의 장비를 조립할수 있다.

미들웨어구성방식

그림 13-8은 의뢰기/봉사기구성방식에서 미들웨어의 역할을 보여 준다. 미들웨어의 정확한 역할은 사용되고 있는 의뢰기/봉사기계산의 형태에 관계된다. 그림 13-5를 다시 보면 응용프로그람기능들을 분산시키는 방법에 따라 여러가지 서로 다른 의뢰기/봉사기방법들이 있다는것을 상기할수 있다. 임의의 경우에 그림 13-8은 필요한 구성방식을 위한 좋은 일반적인 방안을 준다.

미들웨어에는 의뢰기요소와 봉사기요소가 다 있다. 미들웨어의 기본목적은 의뢰기에 있는 응용프로그람 또는 사용자가 봉사기들의 차이점에는 관계없이 봉사기의 여러가지 봉사에 접근하게 하는것이다. 한가지 특정한 응용령역을 보기 위하여 구조화질문언어(SQL)는 국부 및 원격사용자나 응용프로그람으로 관계자료기지에 접근하기 위한 표준수단을 제공한다고 가정한다. 그러나 많은 관계자료기지제작자들은 자기들이 비록 SQL을 지원하여도 자기나름의 확장기능을 SQL에 추가하였다. 이때문에 제작자들은 자기의 제품들을 구별하지만 잠재적인 비호환성을 창조한다.

실례로 특히 간부과를 지원하기 위하여 사용되는 분산체계를 고찰하자. 종업원이름, 주소 등과 같은 기본종업원자료를 Gupta자료기지에 보관하고 반면에 사무원정보는 Oracle자료기지에 보관할수 있다. 간부과에 있는 사용자가 특정한 레코드들에로의 접근을 요구할 때 사용자는 어느 제작자의 자료기지가 필요한 레코드들을 가지고 있는가에는 관심을 가지지 않는다. 미들웨어는 이 서로 다른 체계들에 통일적으로 접근할수 있는 층을 제공한다.

실현의 관점에서보다 오히려 론리적인 관점에서 미들웨어의 역할을 고찰하는것이 좋다. 그림 13-9에서 이러한 관점을 설명한다. 미들웨어는 분산의뢰기/봉사기계산을 실현할수 있게 한다. 완전한 분산체계는 사용자들이 사용할수 있는 응용프로그람과 자원들의

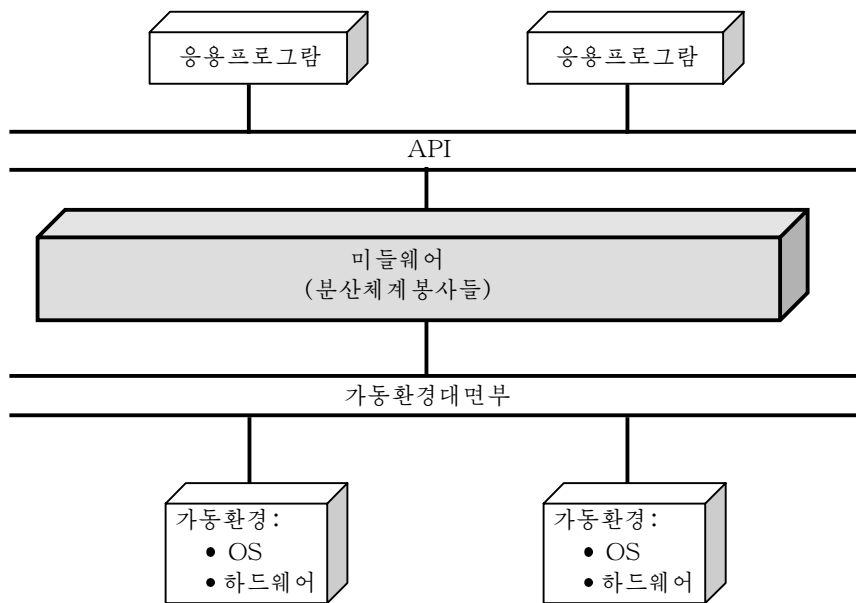



그림 13-9. 미들웨어의 론리적관점[BERN96]

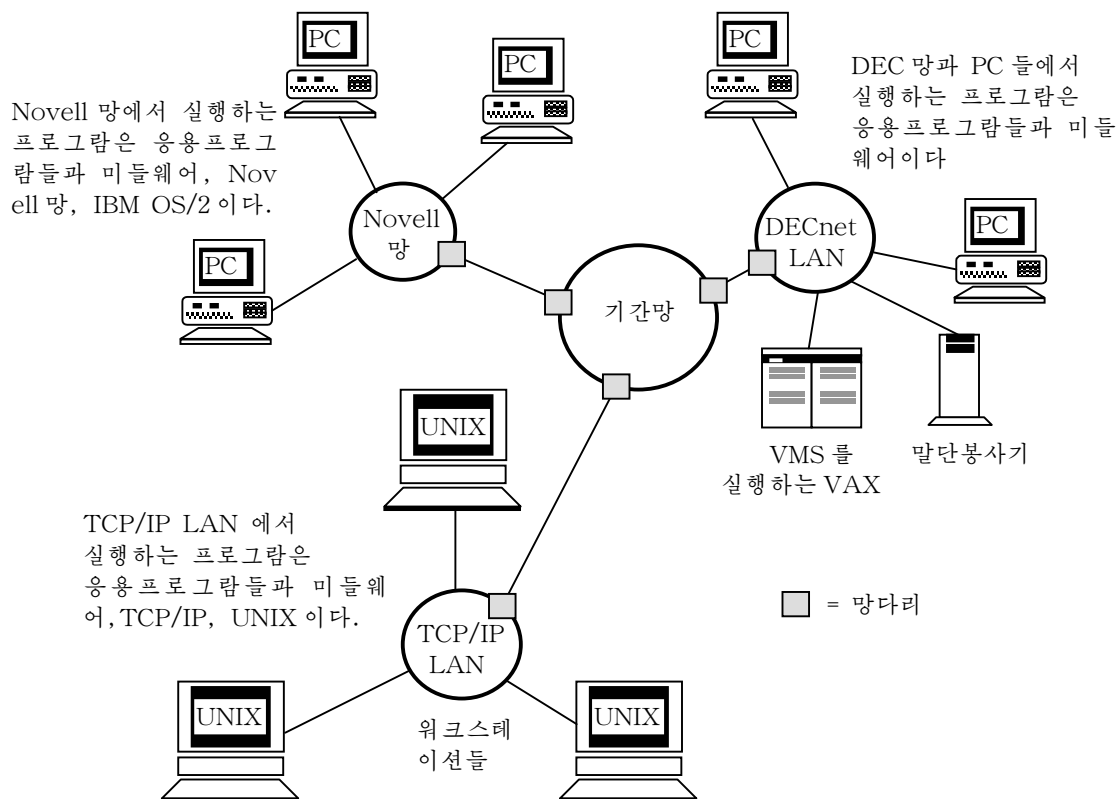



그림 13-10. 미들웨어기능성의 실례

모임으로 볼수 있다. 사용자들은 자료의 위치 또는 실제로는 응용프로그람들의 위치에는 관심이 없다. 모든 응용들은 통일적인 응용프로그람작성대면부(API)에서 동작한다. 모든 의뢰기와 봉사기가동환경을 초월하는 미들웨어는 적당한 봉사기에로 의뢰기요청을 발송해야 한다.

미들웨어를 사용하여 본질적으로 다른 제품들을 어떻게 통합하는가를 보여 주는 실례는 그림 13-10 에서 준 구성이다. 이 경우에 망과 조작체계의 비호환성을 극복하기 위하여 미들웨어를 사용한다. 기간망은 DEC 망과 Novell, TCP/IP 망을 련결한다. 매개 망요소에서 동작하고 있는 미들웨어는 망사용자모두가 임의의 세개 망들에 있는 응용프로그람에는 관심이 없다. 모든 응용들은 통일적인 응용프로그람작성대면부(API)에서 동작한다. 모든 의뢰기와 봉사기가동환경을 초월하는 미들웨어는 적당한 봉사기에로 의뢰기요청을 발송해야 한다.

미들웨어제품의 종류는 매우 많지만 이 제품들은 다음에 취급하는 두 수법들중 하나 즉 통보문넘기기나 원격수속호출에 대표적으로 기초한다. 이 두가지 방법을 다음 두개 절에서 고찰한다.

제 2 절. 분산통보문넘기기

분산처리체계에서는 콤퓨터들이 주기억기를 공유하지 않는다는것 즉 매개는 고립된 콤퓨터체계라는것이 사실이다. 따라서 신호기들과 공통기억기령역의 사용과 같은 공유기억기에 관계되는 처리기내부수법들을 사용할수 없다. 대신에 통보문넘기기에 의거하는 수법들을 사용한다. 이 절과 다음 절에서는 가장 일반적인 방법을 고찰한다. 먼저 단일체계에서 사용되는 통보문들의 간단한 응용을 고찰한다. 다음으로 기본기능인 원격수속호출과 같이 통보문넘기기에 의거하는 개별적인 수법을 고찰한다.

그림 13-11 ㄱ는 의뢰기/봉사기기능을 실현하기 위하여 분산통보문넘기기를 사용하는 것을 보여 준다. 의뢰기프로쎄스는 몇가지 봉사(실례로 파일읽기, 인쇄)를 요구하며 봉사기프로쎄스에 봉사요청을 포함하는 통보문을 보낸다. 봉사기프로쎄스는 요청을 허락하며 응답을 포함하는 통보문을 송신한다. 가장 간단한 형태에서는 다만 두개의 기능 즉 송신과 수신이 필요하다. 송신기능은 목적지를 규정하며 통보문내용을 포함한다. 수신기능은 누가 (《모두》를 포함하여)통보문을 희망하는가를 알려 주며 들어 오는 통보문이 기억되는 완충기를 보장한다.

그림 13-12 는 통보문넘기기의 실현방법을 제시한다. 프로쎄스들은 통보문넘기기모듈의 봉사들을 사용한다. 기본지령들과 파라메터로 봉사요청들을 표현할수 있다. 기본지령은 수행될 기능을 규정하며 자료와 조종정보를 넘기기 위하여 파라메터를 사용한다. 기본지령의 실제형식은 통보문넘기기쏘프트웨어에 관련된다. 그것은 수속호출일수 있으며 또는 그자체가 조작체계의 부분인 프로쎄스의 통보문일수 있다.

송신지령은 통보문을 송신하려는 프로쎄스가 사용한다. 그의 파라메터들은 목적지프로쎄스의 식별자와 통보문의 내용들이다. 통보문넘기기모듈은 이 두 요소들을 포함하는 자료단위를 구축한다. TCP/IP 와 같이 몇가지 통신기능을 사용하여 이 자료를 목적지프로쎄스를 주관하는 기계에 송신한다. 자료단위를 목적체계에서 수신할 때 그것은 통신기능에 의하여 통보문넘기기모듈에 발송된다. 이 모듈은 프로쎄스식별자마당을 조사하고 프로쎄스를 위한 완충기에 통보문을 보관한다.

이 대본에서 수신프로쎄스는 자기의 자발성을 통지하여 완충기령역을 지정하고 수신기본지령으로 통보문넘기기모듈에 알려 통보문들을 수신하여야 한다. 다른 방법은 이러

한 통지를 요구하지 않는다. 대신에 통보문넘기기모듈은 통보문을 수신할 때 몇가지 수신신호로 목적지프로쎄스에 신호한다음 수신된 자료를 공유완충기에서 사용할수 있게 한다.

여러가지 설계문제들이 분산통보문넘기기와 관련되며 이것들은 이 절의 나머지부분에서 설명한다.

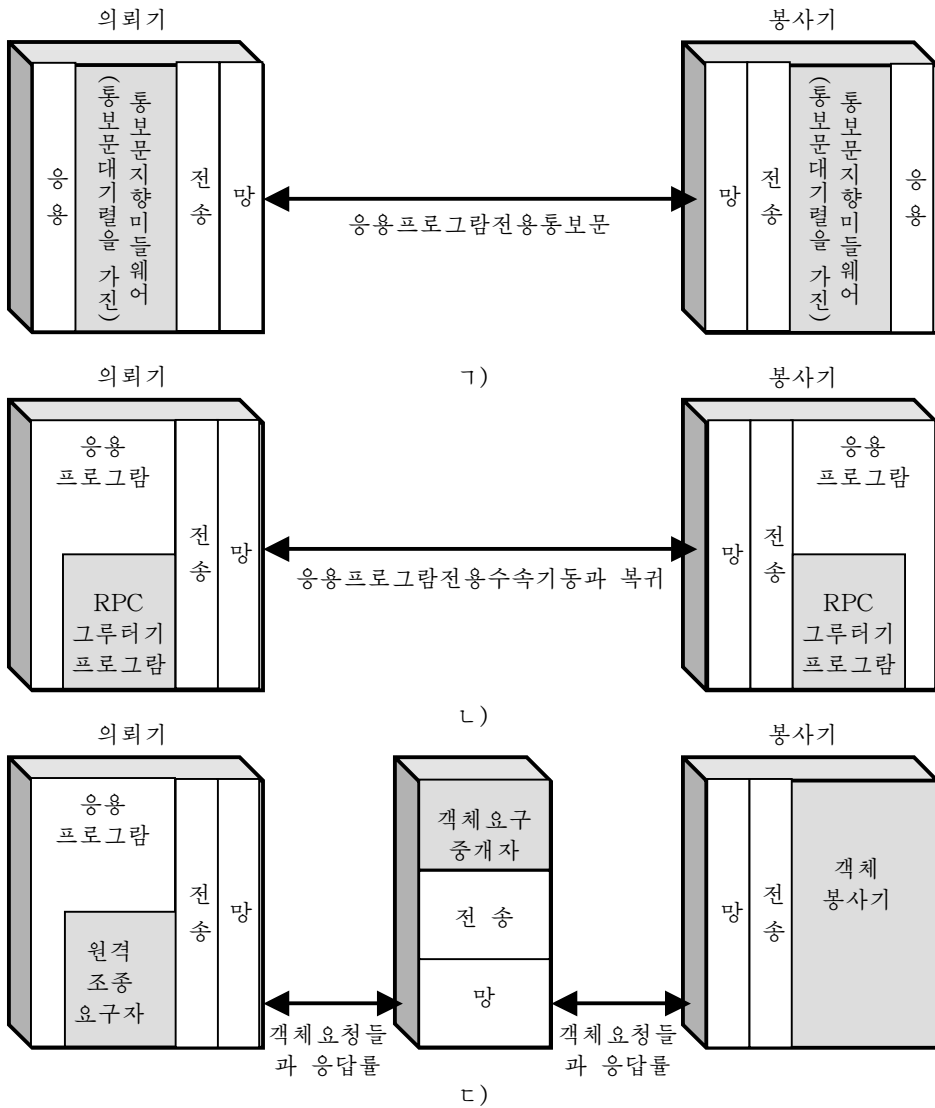



그림 13-11. 미들웨어의 수법
ㄱ-통보문지향미들웨어, ㄴ-원격수속호출, ㄷ-객체요청중개자

믿음성 대 비믿음성

확실한 통보문넘기기기능은 가능한한 전달을 보증하는 기능이다. 이러한 기능은 확실한 전송규약 또는 그와 류사한 론리를 사용할것이며 오유검사, 응답, 재전송, 잘못 배렬된 통보문의 재배렬을 수행할것이다. 전송을 보증하기때문에 송신프로쎄스가 통보문이

전송되였는가를 알게 할 필요는 없다. 그러나 전송이 이미 완료했다는것을 알도록 송신프로쎄스에 거꾸로 응답을 보내는것이 좋다. 어느 경우든 기구가 전송을 달성하지 못하면(실례로 끊임 없는 망고장, 목적지체계의 폭주) 송신프로쎄스는 고장통지를 받는다.

다른 측면에서 통보문넘기기기능은 통신망으로 통보문을 간단히 보낼수 있지만 성공도 고장도 통보하지 않는다. 이 방법은 통보문넘기기기능의 복잡성과 처리, 통신내부처리를 크게 줄인다. 통보문이 전송되였다는 확인을 요구하는 응용프로그람들을 위하여 응용프로그람들자신이 요구를 사용하고 통보문들에 응답하여 요구를 만족시킬수 있다.

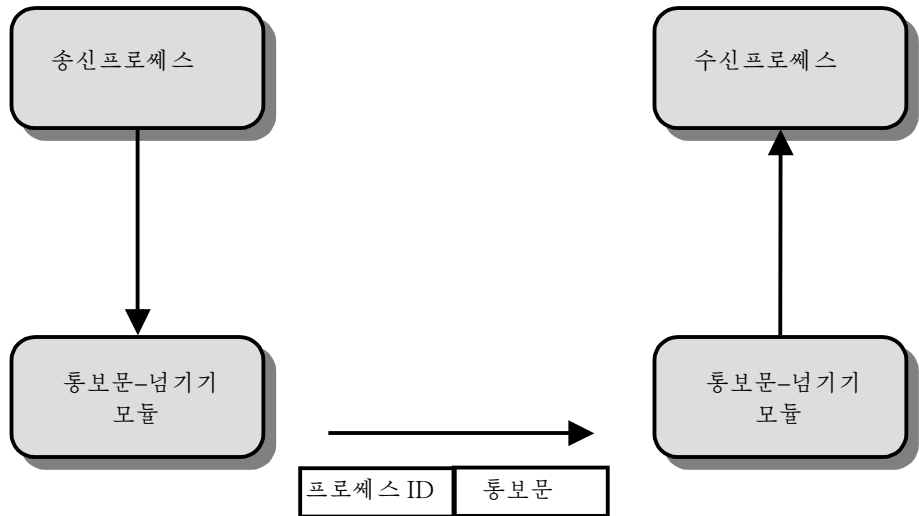



그림 13-12. 통보문넘기기기본지령들

폐색 대 비폐색

비폐색 또는 비동기기본지령들의 경우에 프로쎄스는 송신기본지령 또는 수신기본지령의 출구로 중단되지 않는다. 따라서 프로쎄스가 송신기본지령을 출구하면 조작체계는 통보문을 전송하기 위하여 그것을 대기렬에 넣거나 그 사본이 작성되자마자 프로쎄스에로 조종권을 넘긴다. 사본이 전혀 작성되지 않으면 통보문이 전송되기전이나 전송되는동안에 송신프로쎄스에 의하여 통보문에 가해 진 임의의 변화는 프로쎄스를 위험에 빠뜨린다. 통보문이 다음전송을 위하여 안전한 장소에 전송되였거나 사본이 작성되였을 때 송신프로쎄스는 새치기되고 통보문완충기를 재사용할수 있다는것을 통지 받는다. 이와 류사하게 비폐색수신지령은 그때에 실행하기 시작하는 프로쎄스에 의하여 출구된다. 통보문이 도착하면 프로쎄스는 새치기로 통지를 받거나 주기적으로 상태를 조사할수 있다.

비폐색기본지령들은 프로쎄스들이 통보문넘기기기구를 효률적으로 유연하게 사용하기 위한것이다. 이 방법의 결함은 기본지령들을 사용하는 프로그람들을 검사하고 오유를 수정하기가 힘들다는것이다. 연기할수 없는 동기의존형순차들은 미묘하고 어려운 문제들을 초래할수 있다. 다른 방법은 폐색 또는 동기기본지령들을 사용하는것이다. 폐색송신기본지령은 통보문이 송신된 다음에(비확실한 봉사) 또는 통보문이 송신되고 응답을 수신한 다음에(확실한 봉사) 조종을 송신프로쎄스에 넘긴다. 폐색수신기본지령은 통보문이 배정된 완충기에 놓인 다음에 조종권을 넘긴다.

제 3 절. 원격수속호출

원격수속호출은 기본통보문넘기기의 변종이다. 이것은 분산체계에 통신을 요약하기 위하여 널리 도입된 공통방법이다. 이 수법의 본질은 마치 두 프로그람이 같은 기계에 있는것처럼 다른 기계들에 있는 프로그람들이 간단한 수속호출/복귀의미론들을 사용하여 대화하게 하는것이다. 즉 원격봉사에 접근하기 위하여 수속호출을 사용한다. 이 방법이 보편화된것은 다음과 같은 우점을 가지기때문이다.

1. 수속호출은 광범히 도입사용되고 충분히 알려 진 개념이다.
2. 원격수속호출을 사용하면 원격대면부를 규정된 형태들을 가진 지정된 조작들의 모임으로서 정의할수 있다. 따라서 대면부를 명백히 기록할수 있으며 분산프로그람들의 형태오유를 정적으로 검사할수 있다.

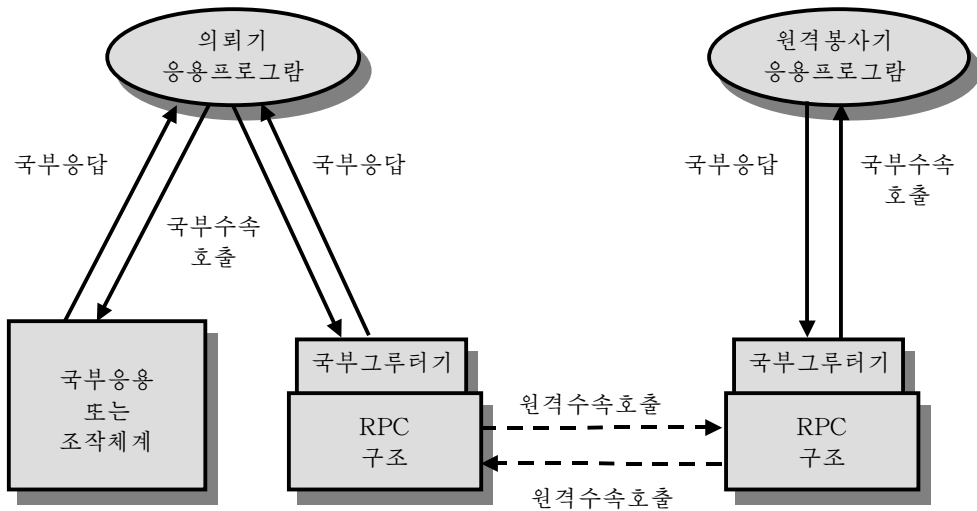



그림 13-13. 원격수속호출기구

3. 표준화되고 정확히 정의된 대면부를 규정하기때문에 응용프로그람을 위한 통신코드는 자동적으로 발생될수 있다.
4. 표준화되고 정확히 정의된 대면부를 규정하기때문에 개발자들은 콤퓨터들과 조작체계들사이로 이동할수 있는 약간 개조되고 재코드화된 의뢰기 및 봉사기모듈들을 작성할수 있다.

원격수속호출기구는 확실히 페색통보문넘기기의 극치로 볼수 있다. 그림 13-11ㄴ는 일반적인 구성방식을 설명하며 그림 13-13은 더 상세한 견해를 준다. 호출하는 프로그람은 기계에 파라메터로 정규수속호출을 한다. 실례로

CALL P(X, Y)

여기서 P = 수속이름
X = 넘겨 지는 인수들
Y = 복귀되는 값들

그것은 다른 콤퓨터에 있는 원격수속을 불러 내려는 사용자에게 투명할수 있고 또 투명하지 않을수 있다. 허위 또는 그루터기수속 P를 호출자의 주소공간에 포함하여야 하거나 호출시에 주소공간에 동적으로 련결하여야 한다. 수속은 호출되는 수속을 식별하는 통보문을 생성하며 파라메터들을 포함한다. 그다음에 그것은 원격체계에 통보문을 보내며 응답을 기다린다. 응답을 수신하면 그루터기수속은 호출하는 프로그람으로 복귀하며 복귀값들을 준다.

원격기계에서 호출된 수속과 관련되는 또다른 그루터기프로그람이 있다. 통보문이 도착하면 조사되며 국부 CALL P(X, Y)이 발생한다. 따라서 원격수속은 국부적으로 호출되므로 파라메터, 탄창상태 등을 어디서 찾아야 하는가에 대한 보통의 가정은 순수한 국부수속호출의 경우와 일치한다.

여러가지 설계문제들이 원격수속호출과 관련되며 그 문제들은 이 절의 나머지부분에서 론의한다.

파라메터넘기기

대부분의 프로그람작성언어들은 파라메터를 값(값에 의한 호출)으로서 넘기거나 값을 포함하는 위치의 지시자(참조에 의한 호출)로서 넘긴다. 값에 의한 호출은 원격수속호출에서 간단히 실현할수 있다. 즉 파라메터는 통보문에 간단히 복사되여 원격체계에 보내 진다. 참조에 의한 호출을 실현하기는 더 힘들다. 매개 객체에는 유일한 체계규모의 지시자가 필요하다. 이것을 실현하기 위하여 노력을 소비할 필요는 없다.

파라메터표현

또다른 하나의 문제는 파라메터들을 어떻게 표현하여 통보문에 반영하는가 하는것이다. 호출되였거나 호출한 프로그람이 같은 조작체계를 가진 같은 형태의 기계들상에서 동일한 프로그람작성언어로 되여 있다면 표현요구문제가 전혀 제기되지 않는다. 이 령역들에서 차이가 있다면 수자와 본문을 표현하는 방법이 다를것이다. 모든 기능을 가진 통신구성방식을 사용하면 표현층으로 이 문제를 처리할수 있다. 그러나 이러한 방식의 내부조작은 대부분의 통신구성방식을 무시하고 자기나름의 기본통신기능들을 제공하는 원격수속호출기능들을 설계하게 하였다. 이 경우에 변환은 원격수속호출기능이 수행한다(실례로 [GIBB87]을 보시오.).

이 문제를 해결하는 가장 좋은 방법은 옹근수, 류동소수점수, 문자, 문자렬과 같이 공통객체들을 위한 표준화된 형식을 취는것이다. 그래야 임의의 기계의 본래파라메터들을 표준화된 표현으로 변환할수 있다.

의뢰기/봉사기맺기

맺기는 원격수속과 호출하는 프로그람사이의 관계가 어떻게 확립되는가를 규정한다. 맺기는 두 응용프로그람이 론리적으로 련결되고 지령들과 자료를 교환할 준비가 되였을 때 이루어 진다.

비지속맺기란 원격수속호출시에 두 프로쎄스사이에 론리적련결을 확립하고 값을 복귀할 때 련결을 취소하는것을 의미한다. 련결은 량쪽의 상태정보의 유지를 요구하기때문에 자원을 소비한다. 이 자원들을 보호하기 위하여 비지속형을 사용한다. 다른한편 련결을 확립하는데 필요한 내부조작으로 하여 같은 호출자가 자주 호출하는 원격수속들에

는 비지속맺기가 적당하지 않다.

지속맺기의 경우에 원격수속호출을 위하여 설정된 련결은 복귀후에 계속 유지된다. 이때 련결은 앞으로의 원격수속호출에 사용될수 있다. 련결에 아무런 작용도 없이 규정된 시간주기가 되면 련결은 취소된다. 원격수속들에 많은 반복호출을 하는 응용프로그람들을 위하여 지속맺기는 론리적인 련결을 유지하며 호출과 복귀절차는 같은 련결을 사용한다.

동기 대 비동기

동기와 비동기원격수속호출의 개념은 페색과 비페색통보문의 개념과 류사하다. 전통적인 원격수속호출은 동기원격수속호출이며 그것은 호출되는 프로그람이 값을 복귀할 때까지 호출하는 프로그람이 기다릴것을 요구한다. 따라서 **동기RPC**는 보조루틴호출처럼 작용한다.

동기RPC는 그 작용을 예측할수 있기때문에 리해와 프로그람작성이 쉽다. 그러나 그것은 분산응용프로그람의 고유한 병렬성을 충분히 살리지 못한다. 이것은 분산응용프로그람이 가질수 있는 대화의 성질을 제한하며 성능을 더 낮춘다.

보다 큰 유연성을 보장하기 위하여 여러가지 **비동기 RPC**기능들이 실현되여 RPC의 친밀성과 단순성을 유지하면서 병렬성을 더 높은 수준에서 달성하였다[ANAN92]. 비동기 RPC들은 호출자를 페색하지 않는다. 즉 응답은 요구에 따라 그리고 필요할 때 수신될수 있다. 따라서 비동기 RPC들은 의뢰기집행이 봉사기기동과 병렬로 국부적으로 진행되도록 한다.

대표적인 비동기RPC의 사용을 본다면 의뢰기가 한번에 많은 요청을 관흐름에 가지고 있도록 봉사기를 몇번이고 기동하게 하는것이다. 이때 매개 요청은 자기의 자료모임을 가진다. 의뢰기와 봉사기의 동기화는 다음의 두 방법중 한가지 방법으로 달성할수 있다.

1. 의뢰기와 봉사기의 상위층의 응용프로그람은 교환을 개시할수 있으며 그다음에 모든 요청된 동작들이 수행된 끝을 검사할수 있다.
2. 의뢰기는 마지막동기 RPC의 앞에 있는 비동기 RPC의 문자렬을 내보낼수 있다. 봉사기는 선행하는 비동기 RPC 들에서 요청한 모든 작업을 완성한후에만 동기 RPC에 응답할것이다.

일부 방법들에서 비동기RPC들은 봉사기에 그 어떤 응답도 요구하지 않으며 봉사기는 응답통보문을 송신할수 없다. 다른 방안들은 응답을 요구하거나 허용하지만 호출자는 응답을 기다리지 않는다.

객체지향기구

객체지향기술이 조작체계설계에 널리 보급됨에 따라 의뢰기/봉사기설계자들은 이 방법을 받아 들이기 시작하였다. 이 방법에서 의뢰기와 봉사기들은 통보문들을 객체들사이에서 앞뒤로 발송한다. 객체통신은 아래준위에 놓여 있는 통보문이나 RPC 기구에 의거하거나 조작체계에서 객체지향성능에 기초하여 직접 개발될수 있다.

봉사를 요구하는 의뢰기는 객체요청중개자에 요청을 보낸다. 객체요구중개자는 망에서 사용가능한 모든 원격봉사의 등록부로서 동작한다(그림 13-11 ㄷ). 중개자는 적당한 객체를 호출하며 임의의 관련자료에 접근한다. 그다음에 원격객체가 요청에 봉사하고 중개자에게 대답들을 주며 중개자는 의뢰기에 그 응답을 되넘긴다.

객체지향방법의 성공은 객체기구의 표준화에 관계된다. 이 령역에서 여러가지 설계안들이 제기되고 있다. 그중의 하나가 객체련결과 매몰(OLE)의 기초인 Microsoft 의 공통객체모형(COM)이며 이 방법은 UNIX 에서의 COM 을 개발한 Digital Equipment 회사의 후원을 받고 있다. Object Manegement Group 가 개발한 경쟁방법은 공통객체요청중개자구성방식(CORBA)인데 이것은 광범한 산업의 후원을 받고 있다. IBM, Apple, Sun 그리고 그밖의 많은 제작자들이 CORBA 방법을 지원하고 있다.

제 4 절. 클라스터

콤퓨터체계설계에서 가장 열띤 새로운 령역의 하나가 클라스터화이다. 클라스터화는 높은 성능과 사용률을 제공하는 방법으로서 대칭다중처리(SMP)의 대안이며 특히 봉사기응용프로그람들에서 인기가 있다. 클라스터는 한대의 기계에 있는 환영(착각)을 창조할수 있는 통합된 계산자원으로서 함께 작업하는 상호련결된 전체 콤퓨터들의 그룹으로 정의할수 있다. 전체 콤퓨터란 말은 클라스터와는 별개의 문제로 독립으로 실행할수 있는 체계를 의미한다. 문헌에서는 클라스터에서의 매개 콤퓨터를 대표적으로 마디라고 한다.

[BREWP97]은 클라스터화가 달성할수 있는 4 가지 리익을 주었다. 이것들을 목적이나 설계요구항목으로 생각할수 있다.

- **절대동시실행성:** 가장 큰 독립기계들의 능력을 훨씬 릉가하는 큰 클라스터들을 창조할수 있다. 클라스터는 수십대 지어 수천대의 기계를 가질수 있으며 매 기계는 다중처리기이다.
- **증분동시실행성:** 클라스터는 새로운 체계들을 조금씩 추가하는 방법으로 구성된다. 따라서 사용자는 현재 가지고 있는 작은 체계를 보다 큰 체계로 교체하는것과 같은 완전한 갱신을 하지 않고 가장 좋은 체계에서 시작하여 요구가 늘어남에 따라 체계를 확장할수 있다.
- **고리용성:** 클라스터의 매개 마디는 독립콤퓨터이기때문에 한개 마디의 고장이 봉사의 실패를 의미하지 않는다. 많은 제품들에서는 쏘프트웨어가 장애극복기능을 자동적으로 조종하고 있다.
- **가격/성능이 낮은 비:** 블로크들을 구축하고 있는 제품을 사용하면 매우 적은 원가로 클라스터가 단일한 큰 기계와 같거나 더 큰 계산능력을 가지게 할수 있다.

클라스터의 구성

문헌에서는 클라스터를 여러가지 다른 방법들로 분류하였다. 가장 간단한 분류법은 클라스터의 콤퓨터가 동일한 디스크의 접근을 공유하는가 안하는가에 따른다. 그림 13-4 ㄱ는 클라스터의 동작을 조정하기 위하여 통보문교환에 사용할수 있는 고속련결방법으로 유일한 상호접속이 이루어 지는 두마디클라스터를 보여 준다. 련결은 다른 비클라스터콤퓨터들이 공유하는 LAN 또는 전용상호접속장치일수도 있다. 후자의 경우에 봉사기클라스터와 원격의뢰기체계사이를 접속하도록 클라스터의 한대 또는 그이상의 콤퓨터들을 LAN 또는 WAN 에 련결할수 있다. 그림에서는 매개 콤퓨터가 다중처리기를 가지고 있는것으로 묘사하였다. 이럴 필요는 없지만 성능과 사용률을 다 높인다.

그림 13-14 에서 제시한 간단한 분류에서 다른 방법은 공유디스크클라스터이다. 이

경우에는 일반적으로 마디들사이에 통보문련결이 되여 있다. 게다가 클라스터에 있는 다중콤퓨터들에 직접 련결한 디스크보조체계가 있다. 이 그림에서 공통디스크보조체계는 RAID 체계이다. 다중콤퓨터들로 하여 달성된 높은 사용률이 고장난 단 하나의 공유디스크때문에 떨어 지지 않도록 클라스터체계에서는 RAID 또는 그와 좀 류사한 중복디스크수법을 일반적으로 사용한다.

기능대안들을 고찰하여 클라스터선택령역에 대한 명백한 그림을 얻을수 있다. Hewlett Packard 의 백서[HP96]는 현재 론의하는 기능선로(표 13-2)에 따르는 쓸모있는 분류를 제공한다.

피동예비(체계)로 알려 진 오랜 공통방법은 1 차 기계의 고장사건을 인계 받기 위하여 대기하고 있는 다른 콤퓨터가 동작하지 않는 동안 한개의 콤퓨터가 모든 처리부하를 조종하게 하는것이다. 기계들을 조정하기 위하여 능동 또는 1 차 체계는 예비기계에 《심장고동》통보문을 주기적으로 송신한다. 이 통보문들의 도착이 정지되면 예비기계는 1 차봉사기가 고장났다고 가정하고 자기가 동작상태에 들어 간다. 이 방법은 사용률을 높이지만 성능을 개선하지 못한다. 더우기 두 체계사이에 교환되는 유일한 정보가 심장고동통보문이고 두 체계가 공통디스크를 공유하지 않는다면 예비기계는 기능예비를 제공하지만 1 차 기계가 관리하는 자료기지에 전혀 접근하지 못한다.

일반적으로 피동예비(체계)를 클라스터라고 하지 않는다. 클라스터라는 용어는 외부세계에 단일체계라는 인상을 주면서 처리를 모두 능동적으로 하려는 다중상호접속콤퓨터들에 적용된다. **능동 2 차(체계)**라는 말은 이러한 구성을 언급할 때 사용된다. 세가

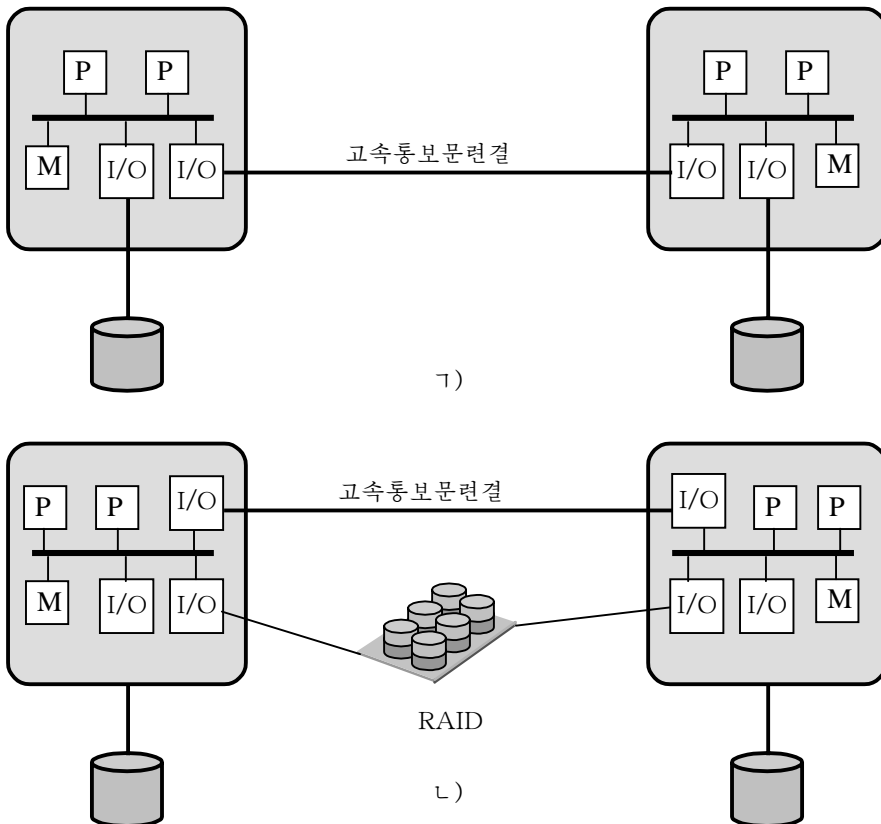



그림 13-14. 클라스터의 구성
ㄱ-공유디스크가 없는 예비봉사기, ㄴ-고유디스크

지 클라스터화방법 즉 개별봉사기, 비공유, 공유기억기를 구분할수 있다.

클라스터화의 한가지 방법은 매개 콤퓨터가 자기의 디스크를 가진 **단독봉사기**이며 체계들사이에 공유되는 디스크가 전혀 없다(그림 13-14 ㄱ). 이러한 구성은 높은 사용률은 물론 높은 성능을 제공한다. 이 경우에 부하의 균형을 보장하고 높은 사용률을 달성하도록 봉사기에 들어 오는 의뢰기요청들을 할당하기 위하여 어떤 종류의 관리 또는 일정작성쏘프트웨어가 필요하다. 즉 고장넘기기능력을 가지는것이 좋다. 고장넘기기능력이란 한 콤퓨터가 응용프로그람을 실행하는중에 고장나면 클라스터에 있는 또다른 콤퓨터가 그 응용프로그람을 가져다 완성할수 있다는것을 말한다. 이것을 실현하자면 매개 체계가 다른 체계의 현재 자료에 접근하도록 상시적으로 체계들사이에서 자료를 복사하여야 한다. 이 자료교환을 위한 내부처리는 성능을 약화시키는 반면에 높은 사용률을 보증한다.

통신내부처리를 줄이기 위하여 대부분의 클라스터들은 현재 공동디스크에 접속된 봉사기들로 구성된다(그림 13-14 ㄴ). 이 방법의 한가지 변종을 간단히 **비공유**라고 한다. 이 방법에서 공통디스크들은 기록권들에 분할되며 매개 기록권은 단일콤퓨터가 소유한다.

표 13-2. 클라스터화방법 : 리익과 제한

클라스터화방법	설명	리익	제한
피동예비체계	2 차봉사기는 1 차봉사기가 고장나는 경우에 계승하여 동작한다	실현하기 쉽다	2 차봉사기는 다른 과제들을 처리하는데 사용할수 없기때문에 원가가 비싸다.
능동 2 차체계	2 차봉사기도 과제처리에 사용할수 있다.	2차봉사기를 과제처리에 사용할수 있기때문에 원가가 감소한다.	복잡성이 증가한다.
단독봉사기	단독봉사기는 자기의 디스크를 가진다. 자료는 1 차봉사기에서 2 차봉사기로 련속적으로 복사된다.	높은 사용률	조작을 모방하기때문에 망과 봉사기내부조작시간이 크다.
디스크에 접속된 봉사기	봉사기들은 동일디스크에 련결되였지만 매개 봉사기는 자기 디스크들을 소유한다. 만일 한개의 봉사기가 고장나면 그 디스크들은 다른 봉사기에 의하여 계속 사용된다.	조작의 모방을 제거하기때문에 망과 봉사기내부조작시간이 줄어 든다.	디스크고장의 위험성을 보상하기 위하여 보충대칭복사 또는 RAID 기능을 요구한다.
봉사기공유디스크	다중봉사기들은 디스크에 대한 접근을 동시에 공유한다.	망과 봉사기내부조작이 작다. 디스크고장에 의하여 발생한 정지시간의 위험성이 감소한다.	잠금관리기쏘프트웨어를 요구한다. 보통 디스크대칭복사 또는 RAID 수법과 함께 사용된다.

그 콤퓨터가 고장나면 어떤 다른 콤퓨터가 고장난 콤퓨터의 기록권에 대한 소유권을 가지도록 클라스터를 재구성하여야 한다.

같은 시간에 동일디스크를 공유하는 다중콤퓨터들을 가질수 있기때문에(**공유디스크** 방법이라고 한다.) 매개 콤퓨터는 모든 디스크의 모든 기록권에 접근한다. 이 방법은 오

직 한대의 콤퓨터가 동시에 자료에 접근할수 있다는것을 보증하기 위하여 일부 잠금기능형태를 사용할것을 요구한다.

조작체계설계문제

클라스터하드웨어구성을 충분히 활용하기 위하여서는 단일체계조작체계를 몇가지 개선하여야 한다.

고장관리

클라스터가 고장을 어떻게 관리하는가는 사용되는 클라스터화방법에 관계된다(표 13-2). 일반적으로 고장을 처리하기 위하여 두가지 방법 즉 사용률이 높은 클라스터와 장애극복력이 있는 클라스터를 사용할수 있다. 사용률이 높은 클라스터는 모든 자원들이 봉사중에 있을것이라는 높은 확률을 제공한다. 체계가 정지하거나 디스크기록권이 잃어 지는것과 같은 고장이 발생하면 진행중에 있던 질문들이 잃어 진다. 임의의 잃어진 질문을 다시 하면 클라스터의 다른 콤퓨터가 그 질문에 봉사할것이다. 그러나 클라스터조작체계는 부분적으로 실행된 트렌잭션들의 상태에 대하여서는 전혀 보증하지 않는다. 이것은 응용프로그람수준에서 조정되여야 한다.

장애극복력이 있는 클라스터는 모든 자원들은 항상 사용할수 있다는것을 보증한다. 이것은 여분의 공유디스크들을 사용하여 그리고 비완료트랜잭션은 취소하고 완성된 트랜잭션만 완료하기 위한 기구들을 사용하여 달성한다.

클라스터에서 고장난 체계로부터 다른 체계에로 응용프로그람과 자료자원들을 절환하는 기능을 **고장넘기기**라고 한다. 그와 관련되는 기능은 일단 고장을 회복하고 원래체계에로 응용들과 자료자원들을 회복하는것인데 이것을 **고장회복**이라고 한다. 고장회복을 자동화할수 있지만 이것은 고장이 완전히 퇴치되였을 때에만 기대할수 있다. 그렇지 않은 경우 자동고장회복은 자원들에 계속 고장을 일으켜 콤퓨터들사이에서 왔다갔다하면서 고장을 퇴치하지 않으면 안되며 결과 성능문제와 회복문제가 발생한다.

부하평형

클라스터는 사용가능한 콤퓨터들사이의 부하균형을 잡기 위한 효과적인 능력을 요구한다. 즉 클라스터가 점차 부하무게를 달수 있을것을 요구한다. 새로운 콤퓨터를 클라스터에 추가하면 부하평형기능은 응용프로그람들의 일정작성에 이 콤퓨터를 자동적으로 포함시킬것이다. 미들웨어기구들은 봉사들이 서로 다른 클라스터성원들에 나타날수 있으며 한 성원으로부터 또다른 성원으로 이동할수 있다는것을 인식할 필요가 있다.

병렬계산

일부 경우들에 클라스터를 효과적으로 사용하자면 쏘프트웨어를 단일응용프로그람에서 병렬로 실행할 필요가 있다. [KAPP00]은 그 문제에 대한 세가지 일반적인 방법을 제시한다.

- **병렬화콤파일러:** 병렬화콤파일러는 콤파일시에 응용프로그람의 어느 부분들이 병렬로 실행될수 있는가를 결정한다. 그다음에 이 부분들을 분해하여 클라스터의 서로다른 콤퓨터들에 할당한다. 성능은 문제의 특성과 콤파일러가 얼마나 잘 설계되였는가에 관계된다.
- **병렬화된 응용프로그람:** 이 방법에서 프로그람작성자는 처음부터 클라스터에서 실행하기 위한 응용프로그람을 작성하며 통보문넘기기를 사용하여 클라스터마

디들사이에서 요구에 따라 자료를 옮긴다. 이것은 프로그람작성자에게 큰 부담을 주지만 일부 응용프로그람에서 클라스터들을 활용하기 위한 가장 좋은 방법일것이다.

- **파라메터계산:** 응용프로그람이 매번 서로 다른 시동조건과 파라메터들을 가지고 여러번 실행되여야 하는 알고리듬이나 프로그람이라면 이 방법을 사용할수 있다. 좋은 실례는 서로 다른 많은 대본들을 실행하고 결과들의 통계적합계를 밝히는 모의모형이다. 이 방법을 더 효과있게 하기 위해서는 정돈된 방법으로 일감들을 조직하고 실행하며 관리하는 파라메터처리도구가 필요하다.

클라스터콤퓨터구성방식

그림 13-15 는 표준적인 클라스터구성방식을 보여 준다. 개별적인 콤퓨터들은 어떤 고속 LAN 이나 스위치하드웨어에 의하여 접속된다. 매개 콤퓨터는 독립적으로 조작할수 있다. 게다가 매개 콤퓨터에는 쏘프트웨어의 미들웨어층이 설치되여 클라스터를 조작할수 있다. 클라스터미들웨어는 사용자에게 **단일체계상**으로 알려 진 통합체계상을 준다. 미들웨어는 부하평형과 개별적인 부분품들에서 발생하는 장애에 따라 높은 사용률을 보장해야 한다. [HWAN99]는 필요한 클라스터미들웨어봉사들과 기능으로서 다음과 같은 것을 제시한다.

- **단일입구점 :** 사용자는 개별적인 콤퓨터가 아니라 클라스터에 가입한다.
- **단일파일계층 :** 사용자는 같은 뿌리등록부밑에 있는 파일등록부들의 단일계층을 고찰한다.
- **단일조종점 :** 클라스터관리와 조종에 사용되는 기정워크스테이션이 있다.
- **단일가상망화 :** 실제 클라스터구성이 다중상호접속망들로 이루어 진다고 해도 임의의 마디는 클라스터의 다른 점에 접근할수 있다. 단일가상망조작이 있다.
- **단일기억기공간 :** 분산공유기억기로 하여 프로그람들은 변수들을 공유할수 있다.
- **단일일감관리체계 :** 클라스터일감일정작성기에서 사용자는 일감을 실행할 주콤퓨터를 규정하지 않고 일감을 제출할수 있다.
- **단일사용자대면부 :** 클라스터에 입구하는 사용자들의 워크스테이션에 관계없이 공통도형대면부는 모든 사용자들을 지원한다.
- **단일입출력공간 :** 임의의 마디는 임의의 입출력주변장치나 디스크장치의 물리적위치를 몰라도 그것들에 원격으로 접근할수 있다.
- **단일프로쎄스공간 :** 통일적인 프로쎄스식별체계가 사용된다. 임의의 마디의 프로쎄스는 원격마디의 임의의 다른 프로쎄스를 생성할수 있거나 그와 통신할수 있다.
- **검사점작성 :** 이 기능은 고장후에 중단점복귀를 위하여 프로쎄스상태와 중간계산결과들을 주기적으로 보관한다.
- **프로쎄스이주 :** 이 기능은 부하평형을 보장한다.

앞에서 설명한 항목들에서 마지막 네개 항목은 클라스터의 사용률을 개선한다. 나머지 항목들은 단일체계상과 관련된다.

그림 13-15 를 다시 보면 클라스터는 병렬실행이 가능한 프로그람들을 효률적으로 실행시키기 위한 쏘프트웨어도구들도 포함한다는것을 알수 있다.

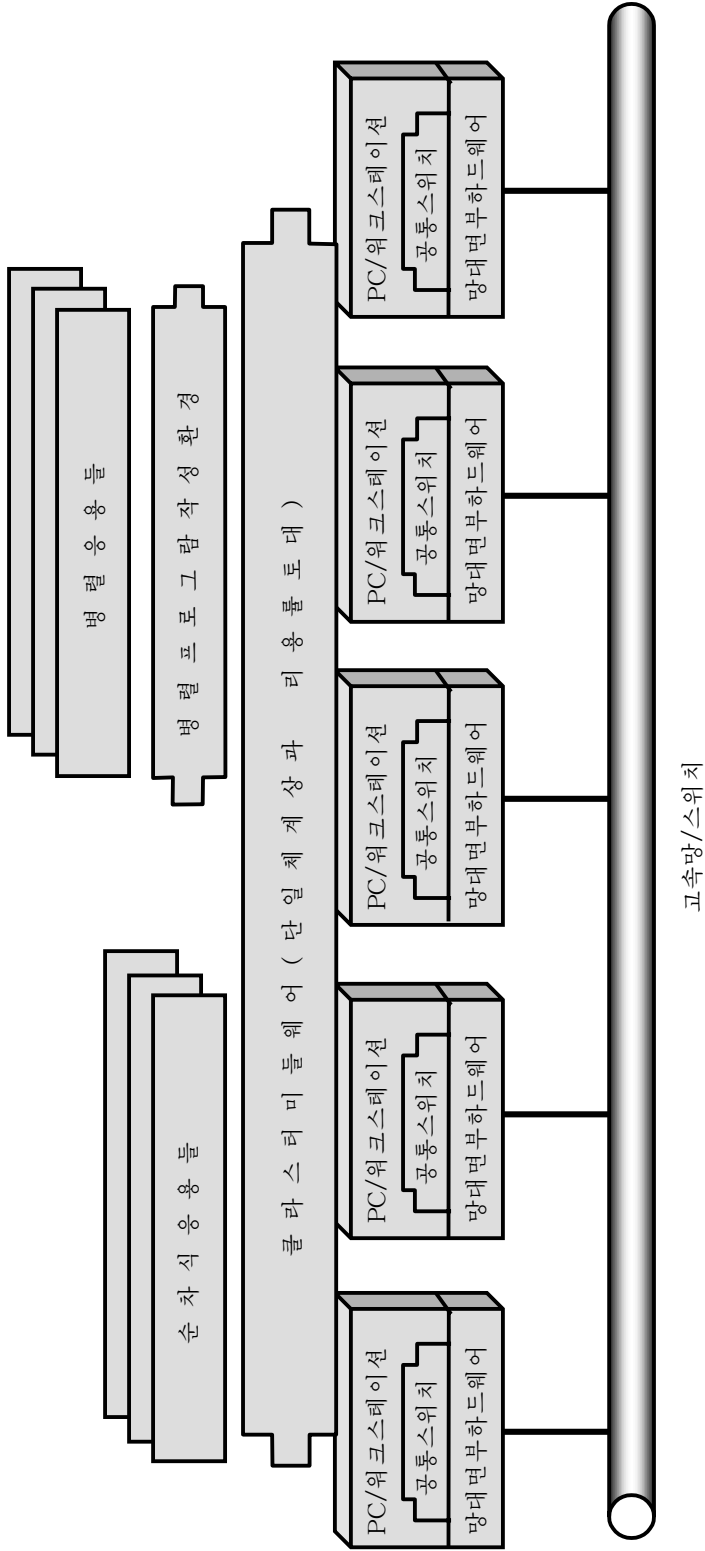



그림 13-15. 클라스터콤퓨터구성방식[BUYYPPA]

클라스터와 SMP의 비교

클라스터와 대칭다중처리기는 모두 다중처리기를 가진 구성을 제공하여 수요가 많은 응용프로그람들을 지원한다. 비록 SMP 는 훨씬 더 오래전부터 나돌고 있지만 두 방법 다 상업적으로 사용할수 있다.

SMP 방법의 기본우점은 클라스터보다 관리하고 구성하기가 더 쉽다는것이다. SMP 는 거의 모든 응용프로그람들이 작성되여 있는 원래의 단일처리기모형에 더욱 가깝다. 단일처리기에서 SMP 로 넘어 가는데 필요한 주요변화는 일정작성기기능에 있다. SMP 의 또다른 우점은 비교되는 클라스터보다 보통 작은 물리적공간을 차지하며 보다 적은 전력을 소비한다는것이다. 마지막 중요한 우점은 SMP 제품들이 매우 믿음직하고 안정하다는것이다.

결국 클라스터가 고성능봉사기시장을 지배할것 같다. 클라스터는 증분 및 절대병행성면에서 SMP 보다 훨씬 우월하다. 클라스터는 또한 체계의 모든 부분품들을 즉시에 매우 여유있게 만들어 낼수 있기때문에 사용률면에서도 우월하다.

제 5 절. Windows 2000 의 클라스터봉사기

Windows 2000(W2K)클라스터봉사기(원래의 코드명은 Wolfpack)는 매개 디스크기록권과 다른 자원들이 단일체계로 동시에 소유되는 비공유클라스터이다.

W2K 클라스터봉사기설계는 다음의 개념들을 사용한다.

- **클라스터봉사 :** 클라스터특유의 모든 동작들을 관리하는 마디의 쏘프트웨어집합
- **자원 :** 클라스터봉사로 관리되는 항목. 모든 자원들은 체계에서 실제자원들을 표현하는 객체들이며 디스크구동기, 망카드와 같은 물리적하드웨어장치와 디스크기록권들, TCP/IP 주소들, 완전한 응용프로그람들, 자료기지들과 같은 론리적항목들을 포함한다.
- **직결 :** 자원이 특정한 마디에 봉사한다면 자원은 마디에 직결되여 있다고 한다.
- **그룹 :** 단일단위로 관리되는 자원들의 집합. 보통 그룹은 특정한 응용프로그람을 수행하는데서와 그 응용프로그람으로 보장되는 봉사에 의뢰기체계를 접속하는데 필요한 모든 요소들을 포함한다.

그룹의 개념은 특별히 중요하다. 그룹은 자원들을 고장넘기기와 부하평형을 위하여 쉽게 관리할수 있는 큰 단위로 결합한다. 그룹을 다른 마디에 이송하는것과 같이 그룹에서 수행된 조작은 그룹에 있는 모든 자원들에 자동적으로 영향을 준다. 자원들은 동적련결서고들(DLLS)로서 실현되며 자원감시기로 관리된다. 자원감시기는 원격수속호출들을 통하여 클라스터봉사와 대화하며 클라스터봉사지령들에 응답하여 자원그룹들을 구성하고 이동시킨다.

그림 13-16 은 단일클라스터체계에서 W2K 클라스터봉사기부분품들과 그들의 관계를 제시한다. **마디관리자**는 클라스터에서 이 마디의 성원자격을 유지해야 한다. 마디관리자는 클라스터의 다른 마디들에 있는 마디관리자들에게 심장고동통보문을 주기적으로 송신한다. 한 마디가 다른 클라스터마디로부터 오는 심장고동통보문을 분실하였다는것을 검출한 경우에 그 마디는 전체 클라스터에 모든 성원들이 통보문을 교환하게 하는 통보문을 발송하여 현재 클라스터성원자격에 대한 그들의 견해를 확증한다. 마디관리자가 응답하지 않으면 그것은 클라스터로부터 제거되며 능동그룹들은 클라스터의 한대이상의 다른 능동마디들에 이송된다.

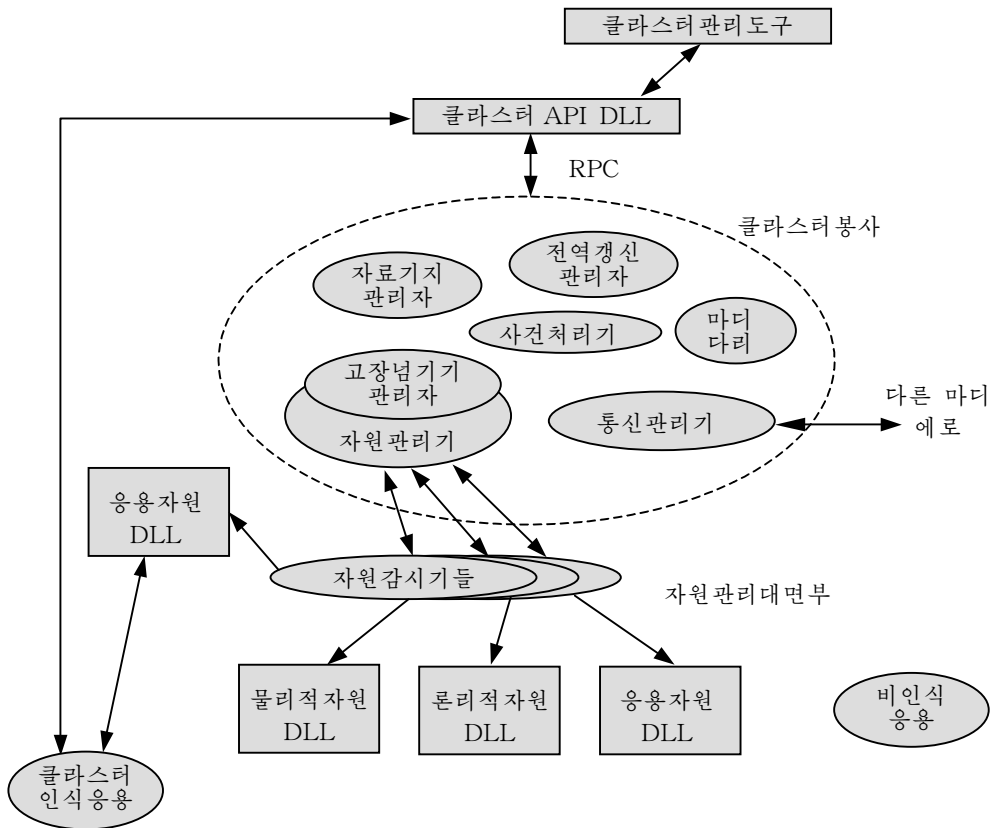



그림 13-16. Windows2000 클라스터봉사기블로크선도[SHOR97]

구성자료기지관리자는 클라스터구성자료기지를 유지한다. 이 자료기지는 자원들, 그룹들, 그룹들의 마디소유권에 대한 정보를 포함하고 있다. 매개 클라스터마디들에 있는 자료기지관리자들은 협동하여 구성정보의 일관성 있는 묘사를 유지한다. 장애극복쏘프트웨어는 전반적인 클라스터구성에서의 변화들이 일관성있게 정확히 수행되였다는것을 확인하는데 사용된다.

자원관리자/고장넘기기관리자는 자원그룹들을 고려하여 모든 결정을 하며 시동, 재설정, 고장넘기기와 같은 적당한 동작들을 개시한다. 고장넘기기가 요구되면 능동마디에 있는 고장넘기기관리자들은 합동하여 고장된 체계로부터 종속되는 능동체계들에로 자원그룹들의 배포를 성과적으로 진행한다. 체계가 고장난후에 정상상태로 되면 고장넘기기관리자는 일부 그룹들을 체계에로 다시 이동시킬것을 결정한다. 특히 임의의 그룹은 우선권이 있는 소유자와 함께 구성될수 있다. 소유자가 실패후에 다시 시작하면 그룹은 재운영조작으로 마디에 다시 옮겨 진다.

사건처리기는 클라스터봉사의 모든 부분품들을 접속하고 공통조직을 조정하며 클라스터봉사초기화를 조종한다. 통신관리자는 클라스터의 다른 모든 마디들과의 통보문교환을 관리한다. 전역갱신관리자는 클라스터봉사에 있는 다른 부분품들이 사용하는 봉사를 제공한다.

제 6 절. SUN 클라스터

Sun 클라스터는 기본 Solaris UNIX 체계에 대한 확장기능들의 모임으로 구축된 분산조작체계이다. 그것은 클라스터단일체계상을 제공한다. 즉 클라스터는 사용자와 응용프로그람들에 Solaris 조작체계를 실행하고 있는 단일콤퓨터처럼 보인다.

그림 13-17 은 Sun 클라스터의 종합적인 구성방식을 보여 준다. 주요한 부분품은 다음과 같다.

- 객체 및 통신지원
- 프로쎄스관리
- 망화
- 전역분산파일체계

객체 및 통신지원

Sun 클라스터의 실현은 객체지향적이다. Sun 클라스터에서 실현되는 객체들과 원격수속호출(RPC)기구를 정의하기 위하여 CORBA 객체모형(부록 2 를 보시오.)을 사용한다. CORBA 대면부정의언어(IDL)를 사용하여 서로 다른 마디들에서 MC 부분품들사이의 대면부를 규정한다. MC 의 요소들은 객체지향언어 C++로 실현된다. 단일한 객체모형과 IDL 의 사용은 마디사이 및 마디밖의 프로쎄스간통신을 위한 기구를 제공한다. 이것들은 모두 핵심부에 실제상 아무런 변화도 요구하지 않고 Solaris 핵심부우에 구축된다.

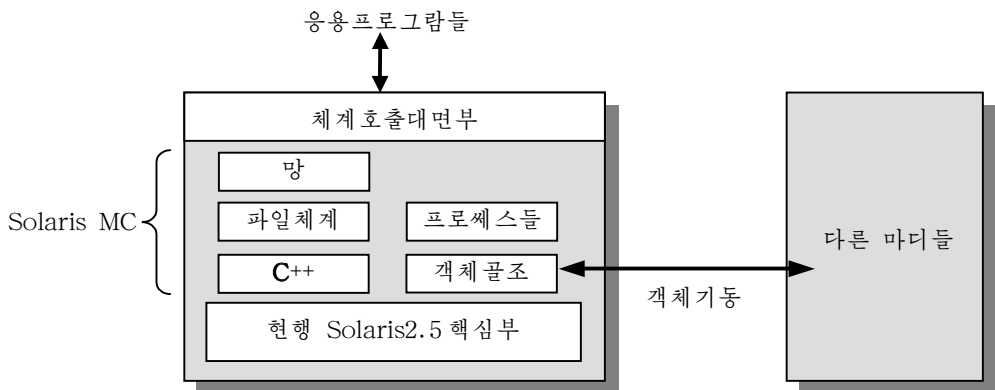



그림 13-17. SUN 클라스터의 구조

프로쎄스관리

프로쎄스관리는 프로쎄스의 위치가 사용자에게 투명하도록 프로쎄스조작들을 확장한다. Sun 클라스터는 클라스터에 있는 매개 프로쎄스가 유일한 식별자를 가지도록 그리고 매개 마디가 매개 프로쎄스의 위치와 상태를 알도록 프로쎄스들의 전역적관점을 유지한다. 프로쎄스이주(제 14 장에서 설명됨)가 가능하다. 프로쎄스는 부하평형의 달성과 고장넘기기를 위하여 자기의 생존기간에 한 마디에서 다른 마디에로 이동할수 있다. 그러나 단일프로쎄스의 스레드들은 같은 마디에 있어야 한다.

망화

Sun 클라스터의 설계자들은 망의 정보흐름을 조정하기 위하여 세가지 방법을 고찰하였다.

1. 모든 망통신규약처리는 단일마디에서 수행된다. 특히 TCP/IP 에 기초한 응용프로그람에서 들어 오는(그리고 나가는) 정보흐름은 들어 오는 정보흐름에 대하여 TCP 와 IP 머리부들을 해석하고 적당한 마디에로 요약된 자료를 발송하며 나가는 통보문에 대하여 다른 마디들로부터의 자료를 TCP/IP 머리부들에 요약하는 망련결마디를 통과한다. 이 방법은 많은 마디들에 적용할수 없으며 따라서 배척된다.
2. 매개 마디에 유일한 IP 주소를 할당하며 매개 마디에서 직접 외부망우에 있는 망규약들을 실행한다. 이 방법의 한가지 난점은 클라스터구성이 외부세계에 더이상 투명하지 않다는것이다. 다른 난점은 실행중의 응용프로그람을 서로 다른 기초망주소를 가지는 다른 마디에로 이동시킬 때 고장넘기기가 어렵다는것이다.
3. 파케트려파기를 사용하여 적당한 마디에로 파케트들을 발송하며 그 마디에서 통신규약처리를 집행한다. 클라스터는 외부에서 단일 IP 주소를 가진 단일봉사기로 보인다. 들어 오는 접속들(의뢰기요구들)은 클라스터의 사용가능한 마디들에 골고루 분담된다. 이것은 Sun 클라스터에서 받아 들인 방법이다.

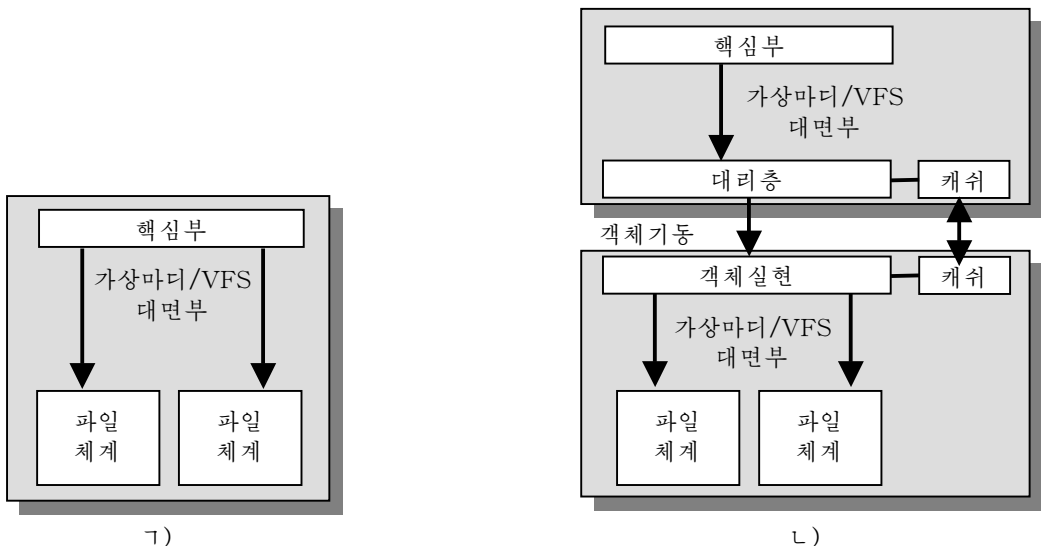



그림 13-18. SUN 클라스터의 파일체계확장
ㄱ-표준 Solaris, ㄴ-Sun 클라스터

Sun 클라스터망화보조체계는 세개의 기본요소를 가진다.

1. 들어 오는 파케트들은 자체에 물리적으로 련결된 망적응기를 가진 마디에 먼저 수신된다. 즉 수신하는 마디는 파케트를 려파하고 그것을 클라스터와 상호접속되여 있는 정확한 목적마디에 송신한다.
2. 나가는 모든 파케트들은 외부망과 물리적접속을 가지고 있는 마디(또는 다중교체마디들중의 하나)에 상호접속된 클라스터전반에로 발송된다.
3. 전역망구성자료기지는 매개 마디의 망정보흐름을 추적하기 위하여 유지된다.

전역파일체계

Sun 클라스터의 가장 중요한 요소는 그림 13-18에 제시된 전역파일체계이다. 이 전역파일체계는 기본 Solaris 구조를 가지는 MC 파일관리와 대조적이다. 두 체계는 가상마디와 가상파일체계의 개념들을 사용하여 구축된다.

Solaris 에서는 가상마디구조를 사용하여 모든 형태의 파일체계에 강력한 범용대면부를 제공한다. 기억기페지들을 프로쎄스의 주소공간에 배치하고 파일체계에로의 접근을 허용하기 위하여 가상마디를 사용한다. 색인마디는 프로쎄스들을 UNIX 파일들로 변환하는데 사용되지만 가상마디는 프로쎄스를 임의의 파일체계형태를 가진 객체로 변환할수 있다. 이 방법에서 체계호출은 조작되는 실체객체를 리해할 필요가 없으며 오직 가상마디대면부를 사용하는 고유객체지향형태호출을 리해하면 된다. 가상마디대면부는 읽기와 쓰기와 같은 범용파일조작지령들을 접수하며 그것들을 객체파일체계에 적합한 동작으로 해석한다. 가상마디를 사용하여 개별적인 파일체계객체들을 서술하는것과 같이 가상파일체계(vfs)구조를 사용하여 전체 파일체계들을 서술한다. vfs 대면부는 전체 파일들을 조작하는 범용지령들을 접수하며 그 지령들을 기본파일체계에 적합한 동작으로 해석한다.

Sun 클라스터에서 전역파일체계는 클라스터에 분산된 파일들에 대한 통일적인 대면부를 제공한다. 프로쎄스는 클라스터의 임의의 장소에 있는 파일을 열수 있으며 모든 마디에 있는 프로쎄스들을 같은 경로명을 사용하여 파일에 배치할수 있다. 전역파일체계를 실현하기 위하여 MC 는 가상마디대면부에서 현행 Solalris 파일체계의 꼭대기에 구축된 대리파일체계를 포함한다. vfs/가상마디조작들은 대리층에 의하여 객체기동들(그림 13-18 ㄴ를 보라.)로 변환된다. 기동된 객체는 체계의 임의의 마디에 있다. 기동된 객체는 기초를 이루는 파일체계에서 국부가상마디/vfs 조작을 수행한다. 이 전역파일환경을 지원하기 위해서는 핵심부도 현재의 파일체계들도 변경되지 말아야 한다.

원격객체기동들의 수를 줄이기 위하여 고속완충화를 사용한다. Sun 클라스터는 파일내용, 등록부정보, 파일속성들의 고속완충을 지원한다.

제 7 절. Beowulf 와 Linux 의 클라스터

1994 년에 NASA 고성능계산 및 통신(HPPC)프로젝트의 발기하에서 Beowulf 프로젝트가 개시되였다. 그의 목표는 최소원가로 당시의 워크스테이션들의 성능을 초월하여 중요한 계산과제들을 수행하기 위한 클라스터화된 PC 들을 연구하는것이였다. 오늘날 Beowulf 방법은 널리 실현되고 있으며 사용할수 있는 가장 중요한 클라스터 기술일것이다.

Beowulf 의 특징

Beowulf 의 기본특징은 다음과 같다[RIDG97].

- 대량판매상품부분품들
- 전용처리기(더우기는 휴식하는 워크스테이션으로부터 오는 환기주기와는 다르다.)

- 전용구내망(LAN 또는 WAN 또는 인터네트에 의한 결합)
- 주문부분품없음
- 다중제작자들로부터의 쉬운 복제
- 부하무게를 달수 있는 입출력
- 자유롭게 사용할수 있는 쏘프트웨어기지
- 최소의 변화로 자유롭게 사용할수 있는 배포계산도구들의 사용
- 설계의 복귀와 사회에로의 개선

Beowulf 쏘프트웨어의 요소들이 많은 서로 다른 가동환경에서 실현되였다고 해도 가장 명백한 기지는 Linux 이며 대부분의 Beowulf 실현들은 Linux 워크스테이션들과 PC 들로 이루어 진 클라스터를 사용한다. 그림 13-19 는 대표적인 구성을 보여 준다. 클라스터는 서로 다른 하드웨어가동환경으로 되여 있으며 모두 Linux 조작체계를 수행하고 있는 많은 워크스테이션들로 구성되여 있다. 매개 워크스테이션에서 2 차기억기는 분산접근(분산공유, 분산가상기억기 또는 다른 용도를 위한)에 사용될수 있다. 클라스터마디들(Linux 통들)은 상용망화수단 대표적으로는 이써네트로 서로 접속된다. 이써네트지원은 단일이써네트스위치 또는 상호접속된 스위치들의 모임형태로 있을수 있다. 상용이써네트제품들은 표준자료속도들(10Mbps, 100Mbps, 1Gbps)로 사용된다.

Beowulf 쏘프트웨어

Beowulf 쏘프트웨어환경은 시장에서 살수 있는 특허사용료가 없는 기본 Linux 배포물들에 대한 부가물로 실현된다. 공개원천 Beowulf 쏘프트웨어의 중요한 원천은 www.beowulf.org 에 있는 Beowulf 싸이트이다. 그러나 수많은 다른 조직들이 역시 무상으로 Beowulf 도구들과 편의프로그람들을 제공한다.

Beowulf 클라스터에 있는 매개 마디는 자기의 Linux 핵심부사본을 실행하며 독립적인 Linux 체계로서 작용할수 있다. Beowulf 클라스터개념을 지원하기 위하여서는 Linux 핵심부에 확장부분을 만들어 개별적인 마디들이 많은 전역이름공간들과 관계를 가지도록 한다. Beowulf 체계쏘프트웨어의 실례들은 다음과 같은것들을 포함한다.

- **Beowulf 분산프로쎄스공간(BPROC) :** 이 제품은 프로쎄스 ID 공간이 클라스터환경에 있는 다중마디들을 포괄하게 하며 다른 마디들에 있는 프로쎄스들을 시동하기 위한 기구도 제공한다. 이 제품의 목적은 Beowulf 클라스터에 단일체계상을 위하여 필요한 기본요소들을 주는것이다. BPROC 는 다른 마디들에 등록하지 않고 클라스터의 통신조종마디의 프로쎄스표에서 모든 원격프로쎄스들을 볼수 있게 함으로써 원격마디들에 있는 프로쎄스들을 시동하기 위한 기구를 제공한다.
- **Beowulf 이써네트통로련결 :** 이것은 여러개의 저원가망들을 더 높은 대역폭을 가지는 단일론리망에로 련결하는 기구이다. 사용하고 있는 단일망대면부에서 수행하여야 할 유일한 보충작업은 사용가능한 장치전송대기렬들에 파케트들을 분산시키는 계산상 단순한 과제이다. 이 방법은 Linux 워크스테이션들에 접속된 다중이써네트들전반에 부하평형을 보장한다.
- **Pvmsync :** 이것은 Beowulf 클라스터에 있는 프로쎄스들을 위한 동기기구들과 공유자료객체들을 주는 프로그람작성환경이다.
- **EnFuzion : 제** 4 절에서 서술한바와 같이 EnFuzion 은 파라메터계산을 하기 위한 도구들의 모임으로 구성되여 있다. 파라메터계산은 매개가 서로 다른 파라메터들 또는 시동조건들을 가지는 수많은 일감들로서 프로그람을 실행한다.

EnFuzsion 은 단일뿌리마디기계에서 로보트사용자들의 모임을 모방한다. 매개 로보트사용자는 클라스터를 형성하는 많은 의뢰기마디기계들중 하나의 기계에 등록할것이다. 매개 일감은 프로그람화된 유일한 각본에 따라 적당한 시동조건 모임을 가지고 실행되도록 설정된다[KAPP00].

요약, 기본용어 및 복습문제

의뢰기/봉사기계산방식은 조직들에서 정보체계들과 망들을 실현하여 생산성을 높이고 개선하기 위한 열쇠이다. 의뢰기/봉사기의 경우에 응용프로그람들은 단일사용자워크스테이션들과 개인용콤퓨터들에 있는 사용자들에 분산된다. 그러나 공유할수 있는 자원들은 모두 의뢰기들에서 사용할수 있는 봉사기체계에 유지된다. 따라서 의뢰기/봉사기 구성방식은 분산계산방식과 집중계산방식의 혼합이다.

대표적으로 의뢰기체계는 사용자가 최소한의 훈련으로 그리고 비교적 쉽게 여러가지 응용프로그람들을 활용할수 있는 도형사용자대면부(GUI)를 보장한다. 봉사기들은 자료기지관리체계들과 같은 공유유틸리티들을 제공한다. 실제응용프로그람은 사용의 편리와 성능을 최량화하기 위한 의도에서 의뢰기와 봉사기에 분해된다.

임의의 분산체계에 요구되는 기본기구는 프로쎄스사이의 통신이다. 두가지 수법이 공통으로 사용된다. 통보문넘기기기능은 단일체계에서 통보문들의 사용을 일반화한다. 같은 종류의 규정들과 동기화규칙들이 적용된다. 다른 방법은 원격수속호출을 사용하는 것이다. 이 수법에 의하여 서로 다른 기계들에 있는 두개의 프로그람은 수속호출/복귀어법과 의미론을 사용하여 대화한다. 호출된 프로그람과 호출하는 프로그람은 둘다 짝패프로그람이 마치 같은 기계상에서 동작하는듯이 행동한다.

클라스터는 한대의 기계가 있다는 환영을 창조할수 있는 통합된 계산자원처럼 함께 작업하는 호상 접속된 완전한 콤퓨터들의 그룹이다. 완전한 콤퓨터라는 말은 클라스터와는 별개로 자기스스로 운영할수 있는 체계를 의미한다.

기본용어

응용프로그람대면부(API)	고장회복	통보문
Beowulf	고장넘기기	미들웨어
의뢰기	살찐 의뢰기	원격수속호출(RPC)
의뢰기/봉사기	파일캐쉬일관성	봉사기
클라스터	도형사용자대면부(GUI)	여윈 의뢰기
분산통보문넘기기		

복습문제

1. 의뢰기/봉사기계산이란 무엇인가?
2. 임의의 형식의 분산자료처리와 의뢰기/봉사기계산을 구별하는것은 무엇인가?
3. 의뢰기/봉사기환경에서 TCP/IP 와 같은 통신구성방식이 노는 역할은 무엇인가?
4. 의뢰기, 봉사기 또는 의뢰기와 봉사기사이에 있는 토막에 응용프로그람을 배치

하기 위한 리론을 론의하시오.

5. 살찐 의뢰기와 여윈 의뢰기란 무엇이며 두 방법이 원리적으로 무엇이 다른가?
6. 살찐 의뢰기와 여윈 의뢰기전략을 위한 찬성과 반대를 제안하시오.
7. 3층의뢰기/봉사기구성방식의 리론적기초를 설명하시오.
8. 미들웨어란 무엇인가?
9. TCP/IP와 같은 표준규격들을 가지고 있는데 왜 미들웨어가 필요한가?
10. 통보문넘기기를 위한 페색 및 비페색기본지령들의 우점과 결함에 대한 목록을 작성하시오.
11. RPC들을 위한 비영구 및 영구맺기의 우점과 결함에 대한 목록을 작성하시오.
12. 동기 및 비동기 RPC들을 위한 우점과 결함에 대한 목록을 작성하시오.
13. 4개의 서로 다른 클라스터화방법들의 목록을 작성하고 간단히 정의하시오.

참 고 문 헌

[SING99]는 이 장에서 론의되는 문제들의 적용범위를 준다. [BERS96]은 의뢰기 및 봉사기에 응용프로그람들을 배정하는 방법과 미들웨어방법들에서 필요한 설계문제들을 기술적으로 론의한다. 이 책은 또한 제품들과 표준화결과들을 론의한다. [REAG00a]와 [REAG00b]에서는 클라스터들의 철저한 처리방법들을 찾아 볼수 있다. 전자는 [RIDG97]에서도 역시 구체적으로 론의하고 있는 Beowulf의 훌륭한 처리방법들을 고찰한다. [STER 99]는 Beowulf의 처리방법을 더 상세히 고찰한다.

[TANE85]는 분산프로쎄스통신과 분산프로쎄스관리를 포함하는 분산조작체계들을 개괄한다. [CHAN90]은 분산통보문넘기기조작체계들을 개괄한다. [TAY90]은 원격수속호출들을 실현하는데서 여러가지 조작체계들이 취한 방법들을 개괄한다.

[PFIS98]은 클라스터에 흥미를 가지는 사람들을 위한 기본도서이다. 이 책은 하드웨어와 쏘프트웨어설계문제들을 취급하고 있으며 클라스터와 SMP를 대비한다. [BUYY99a]와 [BUYY99b]에서는 클라스터들의 철저한 처리방법들을 찾아 볼수 있다. [SHOR97]에서는 W2K 클라스터봉사기를 서술하며 [RAJA00]은 더 상세한 처리방법을 준다. Sun 클라스터는 [SUN99]와 [KHAL96]에서 서술한다.

BERS96 Berson, A. *Clinet/server architecutre.* New York: McGraw-Hill 1996.

BUYY99a Buyya, R. *High Performance Cluster Computing: Architectures and Systems.* Upper Saddle River, NJ: Prentice Hall, 1999.

BUYY99b Buyya, R. *High Performance Cluster Computing: Architectures and Systems.* Upper Saddle River, NJ: Prentice Hall, 1999.

CHAN90 Chandras, R. "Distributed Messgage Passing Operating Systems." *Operating Systems Review,* January 1990.

KHAL96 Khalidi, Y., et al. "Solaris MC: A Multicompter OS." Proceedings, 1996 USENIX Conference, January 1996.

PFIS98 Pfister, G. *In search of clusters.* Upper Saddle River, NJ: Prentice Hall, 1998.

RAJA00 Rajagopal, R. *Introduction to Microsoft Windows NT Cluster Server.* Bocaraton, FL: CRC Press, 2000.
REAG00a Reagan, P. *Client/Server Computing.* Upper Saddle River, NJ: Prentice Hall, 2000.
REAG00b Reagan, P. *Client/Server Network; Design, Operation and Management.* Upper Saddle River, NJ: Prentice Hall, 2000.
RIDG97 Ridge, D., et al. "Beowulf: Harnessing the power of Parallelism in a Pile-of-PCs. "*Proceedings, IEEE Aerospace,*1997.
SHOR97 Short, R.; Gamache, R.; Vert, J.; and Massa, M. "Windows NT Clusters for Availability and Scalability." *Proceedings, COMPCON Spring 97,* February 1997.
SING99 Singh, H. *Progressing to Distributed Multiprocessing.* Upper Saddle River, NJ: Prentice Hall, 1999.
STER99 Sterling, T., et al. *How to Build a Beowulf.* Cambridge. MA:MIT Press, 1999.
SUN99 Sun Microsystems. "Sun Cluster Architecture: A White Paper." *Proceedings. IEEE Computer Society International Workshop on Cluster Computing,* December 1999.
TANE85 Tanenbaum, A., and Renesse, R. "Distributed Operating Systems." Computing Surveys, December 1985.
TAY90 Tay, B., and Ananda, A. "A Survey of Remote Procedure Calls" *Operating Systems Review,* July 1990.

련 습 문 제

1. 클라스터에서 n개의 콤퓨터로 동시에 실행될수 있는 프로그람코드의 비률을 α 라고 하자. 매개 콤퓨터는 서로 다른 파라메터들과 초기조건들의 모임을 사용한다. 나머지코드는 단일처리기로 순차적으로 실행한다고 가정하자. 매개 처리기는 x MIPS의 실행속도를 가진다.

   ㄱ) 프로그람의 배타적인 실행을 위하여 체계를 사용할 때 n, α, x에 관하여 효률적인 MIPS속도를 위한 식을 유도하시오.

   ㄴ) n=16, x=4일 때 40 MIPS의 체계성능을 발휘할수 있는 α의 값을 결정하시오.

2. 응용프로그람이 9대의 콤퓨터로 된 클라스터에서 실행된다. 성능검사프로그람은 이 클라스터에서 T시간 걸렸다. 더우기 T의 25 %는 응용프로그람이 9대의 모든 콤퓨터에서 동시에 실행되는 시간이였다. 그 나머지시간에 응용프로그람은 단일콤퓨터에서 실행하였다.

   ㄱ) 단일콤퓨터에서 프로그람을 실행하는것에 비한 앞에서 말한 조건에서의 효률적인 속도증가를 계산하시오. 앞에서 말한 프로그람에서 병렬처리된(클

라스터방식을 사용하도록 프로그람작성되고 콤파일된) 코드의 비률 α도 계산하시오.

ㄴ) 병렬화된 코드부분에 9대의 콤퓨터가 아니라 18대의 콤퓨터를 효률적으로 사용할수 있다고 가정하자. 이때 달성되는 효률적인 속도증가를 계산하시오.

3. 다음의 FORTRAN 프로그람은 콤퓨터에서 실행되며 병렬판본은 32대의 콤퓨터클라스터에서 실행된다.

```
L1:            DO 10 I=1, 1024
L2:              SUM(I)=0
L3:              DO 20 J=1, I
L4:     20         SUM(I)=SUM(I)+I
L5:     10     CONTINUE
```

2행과 4행은 각각 두개의 기계주기시간을 가지며 처리기와 기억기접근동작을 다 포함한다고 가정하자. 쏘프트웨어고리조종문(행 1, 3, 5)으로 발생하는 내부처리시간과 모든 다른 체계의 내부처리시간, 자원충돌들은 무시하시오.

ㄱ) 단일콤퓨터에서 프로그람의 총 실행시간(기계주기시간으로)은 얼마인가?

ㄴ) 다음과 같이 32대 콤퓨터들에 I반복고리들을 나누시오. 즉 콤퓨터 1은 첫 32개 반복(I=1부터 32까지)을 실행하고 콤퓨터 2는 다음 32개 반복을 실행하는 식으로 나누시오. 실행시간과 부분문제(ㄱ)에 비한 속도증가요인은 무엇인가(J 고리로 명령된 계산작업부하는 콤퓨터들에 균등하게 분배되지 않는것에 주의하시오.)?

ㄷ) 32대 콤퓨터들에서 모든 계산작업부하의 균등한 병렬실행을 쉽게 하기 위하여 병렬화방법을 변화시키시오. 평형부하란 두 고리에 관하여 같은 수의 추가물이 매개 콤퓨터에 할당된다는것을 의미한다.

ㄹ) 32대의 콤퓨터에서 병렬실행한 결과 최소실행시간은 얼마인가? 단일콤퓨터에 비한 속도증가량은 얼마인가?

제 14 장. 분산형프로쎄스의 관리

이 장에서는 분산조작체계에서 사용되는 기본기구들을 고찰한다. 먼저 한 기계에서 다른 기계에로 능동프로쎄스를 이동시키는 프로쎄스이주를 고찰한다. 다음에 서로 다른 체계들에 있는 프로쎄스들이 국부박자로 관리되고 정보교환에서 지연이 있을 때 자기들의 동작을 어떻게 조정하는가 하는 문제를 고찰한다. 마지막으로 분산형프로쎄스관리의 두가지 기본문제 즉 호상배제와 교착을 연구한다.

제 1 절. 프로쎄스의 이주

프로쎄스이주는 목적하는 기계에서 프로쎄스를 실행하기 위하여 한 기계에서 다른 기계에로 프로쎄스의 충분한 상태량을 옮겨 놓는것이다. 이 개념에 대한 관심은 다중망체계들의 부하평형방법들에 대한 연구로부터 발생하였다. 그렇지만 이 개념의 응용은 지금 그 한개 령역을 넘어 확장되고 있다.

지난 시기에는 부하분산에 대한 많은 론문들중에서 일부 론문만이 어떤 기계에 있는 프로쎄스를 선취하여 그것을 다른 기계에서 후에 재가동시킬수 있는 능력이 있는 프로쎄스이주의 실현에 기초하고 있었다. 경험은 선취프로쎄스이주가 원래 예상했던것보다 더 많은 내부처리와 복잡성을 가지지만 가능하다는것을 보여 주었다[ARTS89a]. 이러한 사정은 일부 관찰자들로 하여금 프로쎄스이주가 실용적이지 못하다는 결론을 내리게 하였다. 그러나 이러한 평가는 너무 비관적이라는것이 증명되였다. 시장상품들에서의 실현들을 비롯한 새로운 실현들은 이 령역에 대한 끊임 없는 관심과 새로운 발전을 가져 오게 하였다.

동기

프로쎄스이주는 다음과 같은 몇가지 리유［SMIT88, JUL88］로 하여 분산체계에 필요하다.

- **부하공유**： 무거운 부하를 가진 체계로부터 가벼운 부하를 가진 체계에로 프로쎄스들을 이동시키는것으로 부하의 균형을 맞추어 전체적인 성능을 개선할수 있다. 경험적인 자료는 상당한 성능개선이 가능하다는것을 보여 준다［LELA86, CAB86］. 그러나 부하평형알고리듬의 설계에서는 주의하여야 한다. ［EAGE86］은 분산체계에 필요한 통신이 많으면 많을수록 성능이 더 악화된다는것을 지적한다. 다른 연구들을 참조한 이 문제의 론의는 ［ESKI90］에서 찾아 볼수 있다.
- **통신성능**： 격렬하게 대화하는 프로쎄스들을 대화가 진행되는 동안 같은 마디에 이동시키면 통신비용을 줄일수 있다. 역시 프로쎄스가 자기보다 더 큰 파일 또는 파일묶음에 대한 자료해석을 진행할 때 자료를 프로쎄스가 있는 곳으로 옮기는것보다 반대로 하는것이 유리할것이다.
- **사용성** ： 오래동안 실행하는 프로쎄스들은 사전통고를 할수 있는 장애들에는 아랑곳 없이 또는 예정정지시간보다 더 오래 생존하기 위하여 이동시킬것을 요구할수 있다. 조작체계가 이러한 통고를 한다면 련속하려는 프로쎄스는 다른 체계

에로 이주할수 있거나 얼마간 시간이 지난후에 현재 체계에서 재시동될수 있다.

- **특수능력의 사용**: 프로쎄스는 특정한 마디의 유일한 하드웨어 또는 쏘프트웨어를 사용하기 위하여 이동할수 있다.

프로쎄스이주기구

프로쎄스이주기능을 설계하는데서 몇가지 문제를 강조할 필요가 있다. 그중에는 다음과 같은것들이 있다.

- 누가 이주를 개시하는가?
- 프로쎄스의 무슨 《부분》이 이주되는가?
- 미해결통보문들과 신호들을 어떻게 처리하는가?

이주의 개시

누가 이주를 개시하는가는 이주기구의 목적에 관계된다. 목적이 부하평형이라면 체계부하를 감시하고 있는 조작체계의 모듈이 일반적으로 이주가 발생할 시각을 결정한다. 모듈은 이주될 프로쎄스를 선취하거나 프로쎄스에 신호한다. 이주되는 곳을 결정하기 위하여 모듈은 다른 체계들의 부하패턴들을 감시할수 있도록 다른 체계들의 대등한 모듈들과 통신해야 한다. 목적이 특별한 자원들을 사용하는것이라면 그때 프로쎄스는 요구가 발생하는데 따라 이주할것이다. 후자의 경우 프로쎄스는 분산체계의 존재를 알아야 한다. 전자의 경우 전체 이주기능과 다중체계들의 존재는 프로쎄스에 투명할것이다.

무엇이 이주되는가?

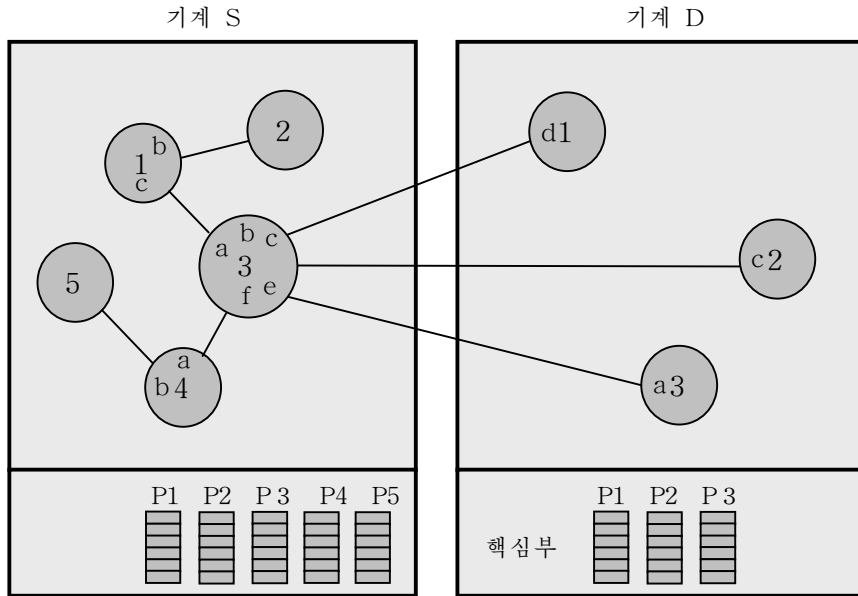
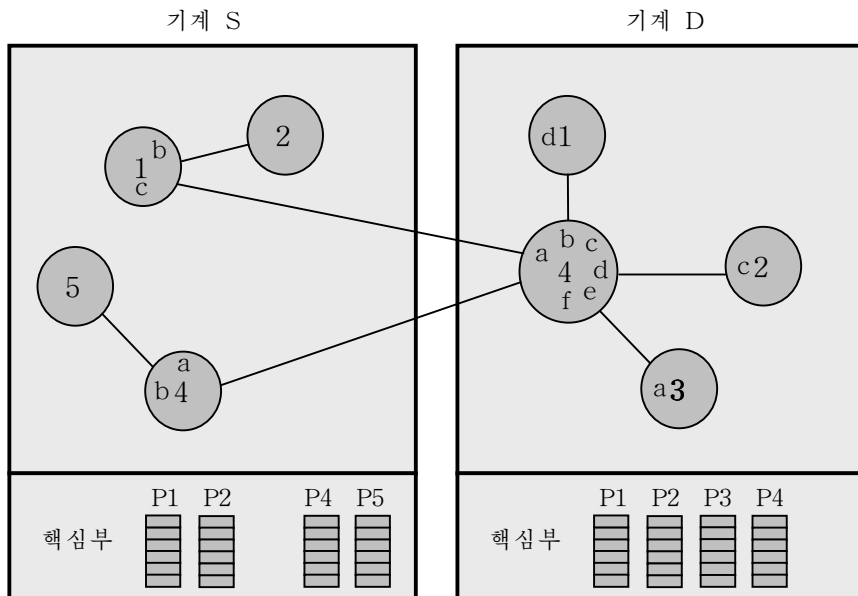
프로쎄스가 이주할 때 원천체계에 있는 프로쎄스는 파괴하고 목적체계에 프로쎄스를 생성해야 한다. 이것은 복사가 아니라 프로쎄스의 이동이다. 따라서 최소한 프로쎄스조종블로크로 구성된 프로쎄스영상을 옮겨야 한다. 더우기 이 프로쎄스와 다른 프로쎄스들사이에 통보문과 신호들을 넘기기 위한 임의의 련결들을 갱신하여야 한다. 그림 14-1은 이러한 내용들을 설명한다. 프로쎄스 3은 기계 S의 밖으로 이주하여 기계 D의 프로쎄스 4로 되였다. （소문자들로 표시된） 프로쎄스들에 보관된 모든 련결식별자들은 전과 같이 남아 있다. 프로쎄스조종블로크들을 이동시키고 련결사영들을 갱신하는 일은 조작체계가 한다. 한 기계의 프로쎄스를 다른 기계에로 이동하는 과정은 이주되는 프로쎄스와 그의 통신상대자들에게 보이지 않는다.

프로쎄스조종블로크의 이동은 간단하다. 성능의 견지에서 난관은 프로쎄스에 할당된 프로쎄스주소공간과 임의의 열린 파일들에 관련된다. 먼저 프로쎄스주소공간을 고찰하자. 그리고 가상기억기방안（토막과 페지화）을 사용하고 있다고 가정하자. 다음의 전략들이 고찰되였다[MILO96].

- **전체공간이송**: 이주시에 전체 주소공간을 이송시킨다. 이것은 확실히 가장 깨끗한 방법이다. 프로쎄스의 흔적을 낡은 체계에 남겨 둘 필요는 전혀 없다. 그러나 주소공간이 매우 크거나 프로쎄스가 주소공간의 대부분을 요구할것 같지 않으면 그때 이 방법은 불필요하게 비용을 소비할것이다. 이주를 위한 초기지출시간은 분단위이다. 검사점작성/재기동기능을 주는 실현들은 이 방법을 사용하는것이 좋다. 왜냐하면 주소공간모두가 국한되면 검사점작성과 재시동을 하기가 더 간단하기때문이다.
- **미리복사**: 프로쎄스는 주소공간이 목적마디에 복사되는 동안 원천마디에서 계속

실행된다. 이 전략은 이주하는 동안에 프로쎄스가 동결되여 실행할수 없는 시간을 감소시킨다.

- **부분공간이송 :** 주기억기에 있거나 변경된 주소공간의 폐지들만을 이송시킨다. 가상주소공간의 임의의 추가적인 블로크들은 요구시에만 이동할것이다. 이것은 이동하는 자료량을 최소로 한다.

그림 14-1. 프로쎄스이주의 실례: ㄱ—이주전, ㄴ—이주후

그러나 원천기계는 페지와 토막 또는 그중의 어느 하나의 표입구점들을 유지하는것으로 프로쎄스의 생존에 계속 관련될것을 요구한다. 또한 원격페지화를 지원할것을 요구한다.

- **참조시복사 :** 이것은 페지들이 참조될 때 참조되는 페지들만을 넘겨 주는 방법으로서 부분공간이송의 변종이다. 이 방법은 프로쎄스이주에 수십㎲~수천㎲ 범위의 가장 적은 초기시간을 요구한다.
- **소각 :** 주기억기에 있는 불결한 페지들을 디스크에 버리면 원천기계의 주기억기에서 프로쎄스의 페지들은 지워 진다. 그때 페지들은 원천마디에 있는 기억기가 아니라 디스크로부터 필요할 때마다 접근된다. 이 전략은 이주되는 프로쎄스의 임의의 페지들이 원천기계의 주기억기에 계속 유지되지 않도록 하며 기억블로크를 즉시에 다른 프로쎄스들에 사용하도록 해방한다.

프로쎄스가 목적기계에 있는동안 주소공간을 많이 사용하지 않으면(실례로 프로쎄스는 다만 파일작업을 위하여 다른 기계에 잠간 왔다가 곧 돌아 오려고 한다.) 마지막 세개 전략들중 한가지 전략이 적합하다. 다른한편 목적기계에 있는동안 결국 많은 주소공간에 접근한다면 이주할 때 주소공간블로크들을 조금씩 이동시키는것은 첫 두가지 전략을 사용하여 주소공간모두를 간단히 이동시키는것보다는 덜 효률적일수 있다.

많은 경우에 많은 비상주주소공간이 요구되겠는지는 사전에 알수 없다. 프로쎄스들이 스레드로 구조화되고 이주의 기본단위가 프로쎄스가 아니라 스레드라면 그때에는 원격페지화에 기초한 전략이 가장 좋을것이다. 참으로 이러한 전략이 대체로 요구된다. 왜냐하면 프로쎄스의 나머지 스레드들은 그대로 남아서 역시 프로쎄스의 주소공간에 대한 접근을 요구하기때문이다. 스레드이주는 Emerlald 조작체계에서 실현된다[JUL89].

류사한 방법들이 열린 파일들의 이동에 적용된다. 파일이 초기에 이주될 프로쎄스와 같은 체계우에 있다면 그리고 파일이 배타적인 접근을 위하여 프로쎄스에 의해 봉쇄되였다면 그때에는 파일을 프로쎄스와 같이 이동시키는것이 합리적이다. 프로쎄스는 오직 림시적으로 이주하며 그것이 돌아올 때까지 파일을 요구하지 않을수도 있다. 그러므로 이주된 프로쎄스에 의하여 접근요구가 있은 다음에만 전체 파일을 이동시키는것이 옳다. 파일이 다중분산프로쎄스들에 의하여 공유되면 파일이동이 없이 파일에로의 분산접근이 유지될것이다.

Sprite 체계(그림 13-11)에서와 같이 고속완충이 허용되면 추가적인 복잡성이 생긴다. 실례로 프로쎄스가 파일을 위하여 파일을 열며 자식프로쎄스를 분기시키고 이주시킨다면 그때 파일은 두개의 서로 다른 주콤퓨터에 쓰기를 위하여 열려 지게 된다. Sprite의 개시일관성알고리듬은 두개의 프로쎄스가 집행하고 있는 기계들에 파일을 고속완충할수 없다고 규정한다[DOUG89, DOUG91].

통보문과 신호

앞에서 제시된 마지막 문제인 통보문들과 신호들에 대한 문제는 이주동작이 진행되는동안 미해결의 통보문들과 신호들을 림시로 기억시키고 그것들을 새로운 목적지에로 보내는 기구를 제공함으로써 해결된다. 모든 미해결통보문들과 신호들이 다 처리된다는것을 보증하기 위하여 어떤 시간동안 초기싸이트에 발송정보를 유지해야 한다.

이주씨나리오

자체이주의 대표적실례로 분산 UNIX 조작체계인 IBM 의 AIX 조작체계[WALK89]에서 사용가능한 기능을 고찰하자. 이와 류사한 기능은 LOCUS 조작체계[POPE85]에서 사용할수 있으며[POPZ85] 사실상 AIX 체계는 LOCUS 조작체계에 기초하고 있다.

사건들은 다음과 같은 순서로 발생한다.

1. 프로쎄스는 자체로 이주할것을 결정하면 목적기계를 선택하고 원격과제통보문을 송신한다. 통보문은 프로쎄스영상부분과 열린 파일정보를 운반한다.
2. 수신측에서는 핵심부봉사기프로쎄스가 자식프로쎄스를 분기시키고 거기에 정보를 준다.
3. 새로운 프로쎄스는 필요에 따라 자료, 환경, 인수 또는 탄창정보들을 불러 와서 자기의 조작을 완성한다. 프로그람본문은 불결하면 복제되거나 깨끗하면 전역파일체계로부터 요구시 페지화된다.
4. 근원프로쎄스는 이주의 완성에 대한 신호를 받는다. 이 프로쎄스는 마지막 완료통보문을 이주한 새로운 프로쎄스에 송신하고 자체로 파괴된다.

다른 프로쎄스가 이주를 개시할 때 이와 류사한 절차로 진행된다. 중요한 차이는 이주되는 프로쎄스가 비집행상태에서 이주될수 있도록 정지되여야 한다는것이다. 실례로 이 절차는 Sprite 가 준수하고 있다[POUG89].

앞에서 서술한 씨나리오에서 이주는 프로쎄스영상을 옮기기 위한 여러 단계를 포함하는 동적동작이다. 자체이주가 아니라 다른 프로쎄스에 의하여 이주가 개시될 때 다른 한가지 방법은 프로쎄스영상과 그의 전체 주소공간을 어떤 파일에 복사하고 프로쎄스를 파괴하며 파일이송기능을 사용하여 다른 기계에로 파일을 복사하는것이다. 그다음에 목적기계에 있는 파일로부터 프로쎄스를 다시 생성하는것이다. [SMIT89]는 이러한 방법을 설명한다.

이주의 협상

프로쎄스이주의 다른 측면은 이주에 대한 결정과 관련된다. 이것은 많은 경우에 단일실체로 결정된다. 실례로 부하평형이 목적이라면 부하평형모듈은 여러가지 기계들에 있는 관련부하를 감시하며 필요에 따라 이주를 수행하여 부하균형을 유지한다. 자체이주를 사용하여 프로쎄스가 특수기능들 또는 더 큰 원격파일들에 접근한다면 프로쎄스는 이주를 자체로 결정할수 있을것이다. 그러나 일부 체계들은 지적된 목적체계가 결정에 참가하도록 한다. 그 목적은 사용자들에게 응답시간을 보장하기 위해서이다. 실례로 워크스테이션의 사용자는 다른 워크스테이션의 프로쎄스들이 자기체계에로 이주한다면 비록 이러한 이주가 더 좋은 전반적균형을 제공하는데 이바지했다고 할지라도 매우 느린 응답시간을 가지게 될수 있다.

협상기구의 실례는 Chalotte[FINK89, ARTS89b]에서 입수한것이다. 이주방책(언제 어느 프로쎄스를 어느 목적지에 이주시키는가)은 장기일정작성과 기억기배정을 담당한 프로쎄스인 Starter편의프로그람의 의무이다. 그러므로 Starter는 이 세개 령역에서 방책을 조정할수 있다. 매개 Starter 프로쎄스는 기계들로 이루어 진 클라스터를 조종할수 있다. Starter 는 매개 기계의 핵심부로부터 정교한 부하통계표를 적당한 때에 공평하게 수신한다.

(그림 14-2 에서 설명한것과 같이) 이주시키기 위한 결정은 두개의 Starter 프로쎄스(원천마디에 있는 프로쎄스와 목적마디에 있는 프로쎄스)가 공동으로 진행하여야 한다. 다음과 같은 단계가 발생한다. 즉

1. 원천체계(S)를 조종하는 Starter 는 프로쎄스 P 가 특정한 목적체계 (D)로 이주될것이라는것을 결정한다. 그것은 D 의 Starter 에 통보문을 보내여 이동을 요구한다.
2. D 의 Starter 가 프로쎄스를 수신할 준비가 되여 있으면 긍정응답을 거꾸로 송신한다.
3. S 의 Starter 는 봉사호출 (Starter 가 S 에서 수행되고 있다면) 또는 기계 S 의 Kewrn Job(KJ)에로의 통보문(Starter 가 다른 기계에서 수행되고 있다면)을 통하여 S 의 핵심부에 이 결정을 통보한다. KJ 는 원격들로부터의 통보문들을 봉사호출들로 변환하는데 사용되는 프로쎄스이다.
4. 그다음에 S 에 있는 핵심부는 프로쎄스를 D 에 송신할것을 제안한다. 제안은 나이, 처리기, 통신부하와 같은 P 에 대한 통계표들을 포함한다.
5. D 의 자원이 불충분하면 D 는 제안을 거절할수 있다. 그렇지 않은 경우에 D 에 있는 핵심부는 자기를 조종하는 Starter 에 제안을 중계한다. 중계는 S 로부터의 제안과 같은 정보를 포함한다.

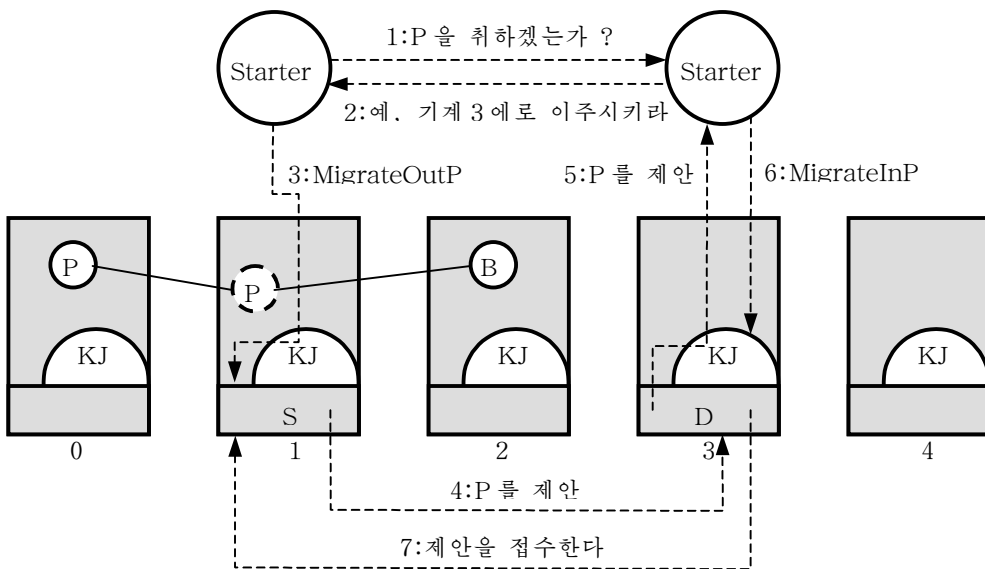



그림 14-2. 프로쎄스이주의 협상

6. Starter 의 방책결정은 MigrateIn 호출로 D 에 통보된다.
7. D 는 교착과 흐름조종문제들을 회피하기 위하여 필요한 자원들을 확보해 둔 다음 S 에 허락신호를 송신한다.

그림 14-2 는 P 와 련결되여 있는 두개의 다른 프로쎄스 A 와 B 를 보여 준다. 앞에서 서술한 단계에 따르면 S 가 있는 기계 1 은 기계 0 과 2 에 련결갱신통보문을 송신하여 A 와 B 로부터 P 까지 련결들을 유지하여야 한다. 련결갱신통보문은 P 가 유지하고 있는 매개 새로운 련결주소를 알려 주며 동기화목적을 위하여 통지를 받은 핵심부들은 련결갱신통보문에 응답한다. 그후에 P 의 임의의 련결들중의 어느 하나의 련결로 P 에 송신된 통보문은 직접 D 에로 송신될것이다. 통보문들은 방금 서술한 단계에 따라 동시에 교환될수 있다. 결국 단계 7 이후에 그리고 모든 련결들이 갱신된후에 S 는 P 의 상태모두를

단일통보문에 수집하여 D 에 송신한다.

기계 4 도 역시 Charlotte 를 실행하고 있지만 이 이주에는 필요 없다. 그러므로 기계 4는 이 실례에서 다른 체계들과 전혀 통신을 진행하지 않는다.

퇴거

협상기구에 의해 목적체계는 자기에로의 프로쎄스이주를 거절할수 있다. 게다가 그것은 역시 체계가 자기에게 이주된 프로쎄스를 퇴거시키게 하는데 쓸모 있다. 실례로 작업기가 무부하상태이면 한개이상의 프로쎄스를 워크스테이션에 이주시킬수 있다. 일단 그 워크스테이션의 사용자가 능동으로 되면 이주된 프로쎄스를 퇴거시켜 적당한 응답시간을 보장할 필요가 있다.

퇴거가능성에 대한 실례는 Sprite[DOUG89]에서 입수한것이다. 워크스테이션조작체계인 Sprite 에서 매개 프로쎄스는 자기의 생존기간 내내 단일주콤퓨터에서 실행한다. 이 주콤퓨터를 프로쎄스의 고향마디라고 한다. 프로쎄스가 이주되면 그것은 목적기계에서 외래프로쎄스로 된다. 임의의 시각에 목적기계는 외래프로쎄스를 퇴거시킬수 있다. 그러면 왜래프로쎄스는 자기의 고향마디에로 다시 이주된다.

Sprite 퇴거기구의 요소들은 다음과 같은 단계로 동작한다. 즉

1. 감시기프로쎄스는 언제 새로운 외래프로쎄스들을 받아 들여야 하는가를 결정하기 위하여 현재부하를 조사한다. 감시기는 워크스테이션의 조종탁에서 어떤 동작을 검출하면 매개 외래프로쎄스의 퇴거수속을 개시한다.
2. 프로쎄스는 퇴거되여 자기의 고향마디에로 다시 이주된다. 프로쎄스가 또 다른 마디를 사용할수 있으면 다시 이주될수 있다.
3. 모든 프로쎄스를 퇴거시키는데 얼마간 시간이 걸린다 하더라도 퇴거를 위하여 표식된 모든 프로쎄스들은 즉시에 중단된다. 퇴거되는 프로쎄스를 퇴거를 기다리는 동안 계속 집행하면 프로쎄스가 동결되여 있는 시간을 줄일수 있지만 퇴거가 진행중에 있는 동안 주콤퓨터에서 사용할수 있는 처리능력은 감소할것이다.
4. 퇴거하는 프로쎄스의 전체 주소공간이 고향마디에로 이동한다. 프로쎄스를 퇴거시켜 그것을 다시 자기의 고향마디에로 이주시키는 시간은 참조된 그의 이전 외래주콤퓨터에서 퇴거하는 프로쎄스의 기억기영상을 회복하는것으로 실제상 줄어 들것이다. 그러나 이것으로 하여 외래주콤퓨터가 자원들을 소비하여야 하며 필요이상 더 오랜 시간주기동안 퇴거하는 프로쎄스가 신용봉사를 하여야 한다.

선취이송과 비선취이송

이 절에서는 부분적으로 실행된 프로쎄스 또는 적어도 생성이 완성된 프로쎄스의 이송을 동반하는 선취이주에 대하여 검토하였다. 더 간단한 기능은 집행을 시작하지 않았으며 따라서 프로쎄스상태의 이송을 요구하지 않는 프로쎄스들만을 포함하는 비선취프로쎄스이송이다. 두가지 이송형태에서 프로쎄스가 실행할 환경에 대한 정보는 원격마디에 이송되여야 한다. 이 정보는 사용자의 현재 작업등록부, 프로쎄스에서 물려 받은 특권들 그리고 파일서술과 같은 계승된 자원들을 포함할수 있다.

비선취이주는 부하평형에 사용할수 있다(실례로 [SHIV92]를 보라.). 그것은 완전한 프로쎄스이주의 내부처리를 회피한다는 우점을 가진다. 결함은 이러한 기구가 부하분배에서의 갑작스러운 변화들에 잘 반응하지 못한다는것이다.

제 2 절. 분산형전역상태

전역상태와 분산순시상기록

호상배제, 교착, 고갈과 같은 밀접히 련결된 체계에서 맞다드는 모든 병행문제들은 역시 분산체계에서도 맞다들게 된다. 이러한 령역들에서의 설계전략들은 체계의 전역상태가 전혀 없다는 사실로 하여 복잡해 진다. 즉 조작체계 또는 임의의 프로쎄스가 분산체계의 모든 프로쎄스의 현재 상태를 아는것은 불가능하다. 프로쎄스는 기억기에 있는 프로쎄스조종블로크들에 접근하여 국부체계에 있는 모든 프로쎄스의 현재상태만을 알수 있다. 프로쎄스는 원격에 대하여서는 통보문으로 수신된 상태정보만을 알수 있다. 그 상태정보는 과거의 언젠가의 원격상태를 표시한다. 이것은 천문학에서의 사정과 류사하다. 즉 먼거리에 있는 별 또는 은하에 대한 지식은 멀리 있는 객체로부터 도달하는 빛과 전자기파들로 이루어 지며 이 파들은 과거의 언젠가의 객체의 영상을 준다. 실례로 5 광년 거리에 있는 객체에 대한 지식은 5 년전의것이다.

분산체계들의 특징으로 하여 생긴 시간지연은 병행과 관련이 있는 모든 문제들을 복잡하게 만든다. 이것을 설명하기 위하여 [ANDR90]에서 준 실례를 제시한다. 프로쎄스/사건그라프(그림 14-3 과 14-4)를 사용하여 문제를 설명한다. 이 그라프에서 수평선은 매개 프로쎄스의 시간축을 표시한다. 선우의 점은 사건에 대응한다(실례로 내부프로쎄스사건, 통보문송신, 통보문수신). 점을 둘러 싸고 있는 상자는 그 점에서 취한 국부프로쎄스상태의 순시상기록을 표시한다. 화살표시는 두 프로쎄스사이의 통보문을 표시한다.

실례에서 개체는 은행의 두개 지점에 분산된 은행게시를 가진다. 구매자게시의 합계

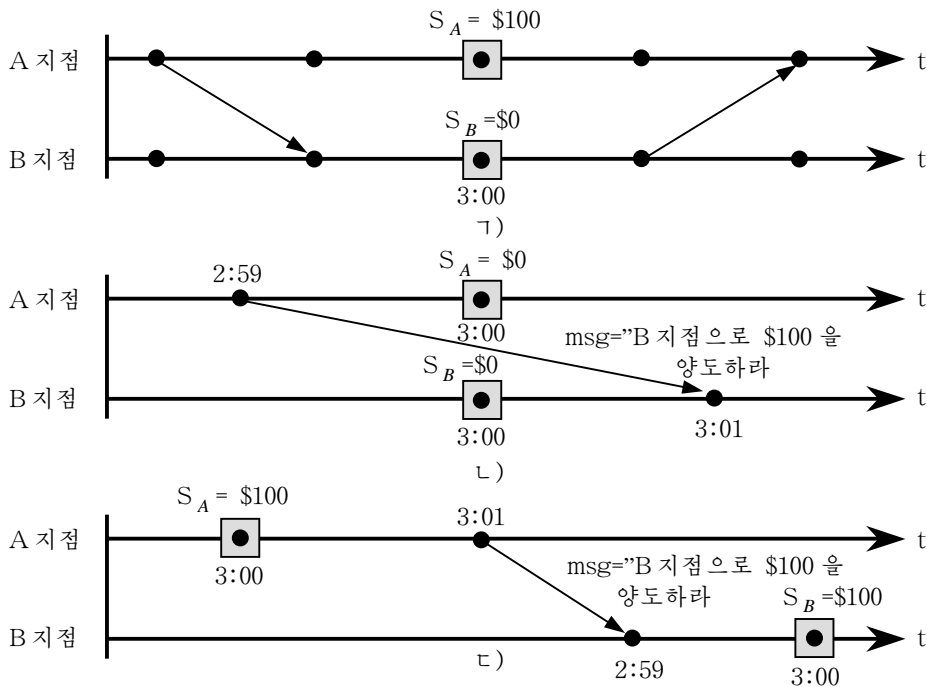



그림 14-3. 전역상태를 결정하는 실례
ㄱ- 총액=$100, ㄴ-총액=$0, ㄷ-총액=$200

를 결정하기 위하여 은행은 매개 지점에서의 총액을 결정하여야 한다. 결정은 정확히 오후 3시에 된다고 가정하자. 그림 14-3 ㄱ는 련합된 게시에서 100$의 잔액을 발견한 구체례를 보여 준다. 그러나 그림 14-3 ㄴ에서의 상태도 가능하다. 여기서 A 지점에서의 잔액은 관찰하는 그 시간에 B 지점으로 이동중이다. 결과는 0.00$으로서 잘못된 읽기이다. 이 특수한 문제는 관찰하는 시간에 이동중인 모든 통보문들을 조사하는것으로 해결될수 있다. A 지점은 양도자료의 목적지신원과 함께 모든 양도자료들의 레코드를 게시에 로출시키지 않는다. 그러므로 A 지점의 게시의 《상태》에 현재잔액과 양도자료들의 레코드를 포함할수 있다. 두개 게시를 조사할 때 관측자는 B 지점에 있는 구매자의 게시로 향한 A지점을 떠난 양도자료를 발견한다. 총액은 B지점에 아직 도착하지 않았기때문에 총잔액에 첨부된다. 임의의 총액은 수신게시에서 단 한번만 계산된다.

이 전략은 그림 14-3 ㄷ에서 보여 준바와 같이 간단명료하지 않다. 이 실례에서 두 지점의 시계는 완전히 동기되여 있지 않다. A 지점에서 오후 3시에 구매자게시의 상태는 100$의 잔액을 가리키고 있다. 그러나 이 총액은 A 지점에 있는 시계에 따라서 3 시 01 분에 B 에 계속하여 양도되지만 B 의 시계에 따라서 2 시 59 분에 B 에 도착한다. 그러므로 총액은 3 시:00 분 관측에서 두번 계산된다.

직면한 문제를 리해하고 해결방법을 공식화하기 위하여 다음의 용어를 정의하자.

- **통로 :** 두개의 프로쎄스가 통보문을 교환한다면 그것들사이에 통로가 존재한다. 통보문을 이송하는 경로 또는 수단으로서 통로를 생각할수 있다. 편의를 위하여 통로들은 한 방향통로라고 본다. 따라서 두개 프로쎄스가 통보문들을 교환할 때 통로가 통보문이송의 매개 방향에 하나씩 필요하다고 보면 두개 통로가 요구된다.
- **상태 :** 프로쎄스의 상태는 프로쎄스에 련결된 통로들을 통하여 송신 및 수신된 통보문들의 순서이다.
- **순시상기록 :** 순시상기록은 프로쎄스의 상태를 기록한다. 매개 순시상기록은 마지막 순시상기록이래 모든 통로들로 송신 및 수신된 통보문들에 대한 기록을 포함한다.
- **전역상태 :** 모든 프로쎄스들의 련합된 상태
- **분산순시상기록 :** 매개 프로쎄스에 하나씩 있는 순시상기록의 집합

문제는 통보문이송과 관련된 시간지연때문에 진짜 전역상태를 결정할수 없다는것이다. 모든 프로쎄스들로부터 순시상기록들을 수집하는것으로 전역상태를 정의할수 있다. 실례로 순시상기록을 취하는 시간에 그림 14-4 ㄱ의 전역상태는 〈A, B〉통로와 〈A, C〉통로, 〈C, A〉통로에서 이송중인 통보문을 보여 주고 있다. 통보문 2와 4는 적당히 표시되였지만 통보문 3 은 그렇지 못하다. 분산순시상기록은 이 통보문이 수신되였지만 아직 송신되지 않았다는것을 가리킨다.

우리는 분산순시상기록이 일관성 있는 전역상태를 기록할것을 희망한다. 전역상태는 매개 프로쎄스상태에 대하여 통보문의 접수를 기록한다면 일관성 있는것으로 되며 이때 통보문의 송신은 통보문을 송신한 프로쎄스의 처리상태로 기록된다. 그림 14-4 ㄴ는 한가지 실례를 보여 준다. 일관성이 없는 전역상태는 프로쎄스가 통보문의 접수를 기록하였지만 대응하는 송신프로쎄스가 통보문을 송신하였다는것을 기록하지 않았을 때 발생한다 (그림 14-4 ㄱ).

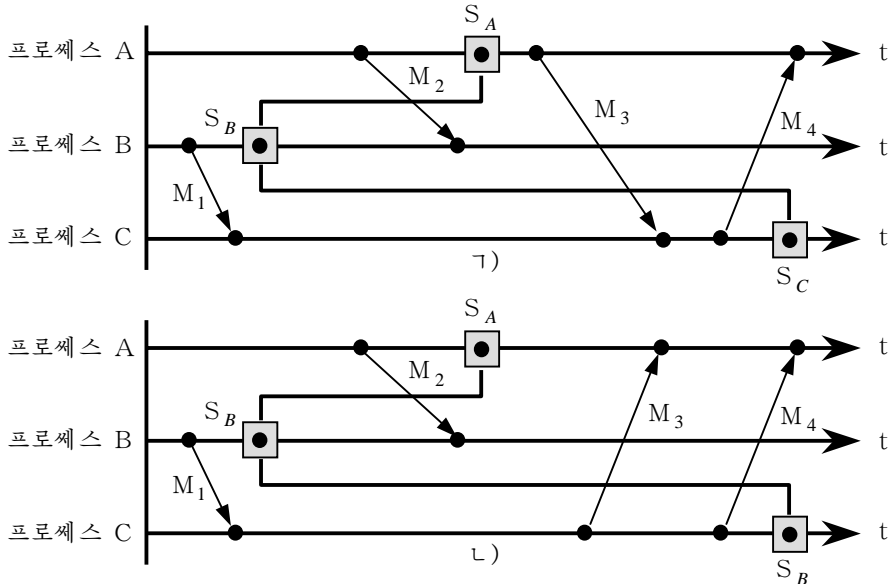



그림 14-4. 일관성 없는 전역상태와 일관성 있는 전역상태
ㄱ-일관성 없는 전역상태, ㄴ-일관성 있는 전역상태

분산순시상기록알고리듬

일관성 있는 전역상태를 기록하는 분산순시상기록알고리듬은 [CHAN85]에 서술되였다. 이 알고리듬은 통보문들이 송신되는 순서로 송달되며 전혀 잃어 지지 않는다고 가정한다. 믿음직한 운반층규약(실례로 TCP)은 이 요구들을 만족시킨다. 알고리듬은 **표식자**라고 하는 특수한 조종통보문을 사용한다.

어떤 프로쎄스는 임의의 더 많은 통보문들을 송신하기전에 자기의 상태를 기록하고 모든 나가는 통로들에 표식자를 송신하는것으로 알고리듬을 개시한다. 그다음 매개 프로쎄스 P는 다음과 같이 진행한다. 표식자 (프로쎄스 Q로부터의 표식자라고 하자.)의 첫 접수를 하자마자 수신처리는 다음의 동작을 수행한다.

1. P는 국부상태 SP를 기록한다.
2. P는 Q에서 P로 들어 오는 통로의 상태를 빈것으로 기록한다.
3. P는 모든 나가는 통로들을 통하여 자기의 모든 이웃들에 표식자를 전달한다.

이 단계들은 자동적으로 수행되여야 한다. 즉 3단계 모두가 수행될 때까지 P는 그 어떤 통보문도 송신 또는 수신할수 없다.

자기의 상태를 기록한후 임의의 시간에 P는 다른 들어 오는 통로(처리기 R로부터라고 하자.)로부터 표식자를 수신하면 다음과 같이 동작한다.

1. P는 P가 자기의 국부상태 SP를 기록한 시각부터 R 표식자를 수신한 시작까지 수신한 프로쎄스들의 순서로서 R에서 P에로의 통로상태를 기록한다.

알고리듬은 일단 매개 표식자가 들어 오는 통로를 통하여 수신되였으면 프로쎄스에서 끝난다.

[ANDRFO]은 알고리듬에 대하여 다음의 관찰을 진행한다.

1. 임의의 프로쎄스는 표식자를 송신하는것으로 알고리듬을 시작할수 있다. 사실상 여러개의 마디들이 독립적으로 결심하여 상태를 기록할것이며 알고리듬은 여전히 계속될것이다.
2. 알고리듬은 매개 통보문(표식자통보들을 포함하여)이 제한된 시간에 송달된다면 제한된 시간에 끝날것이다.

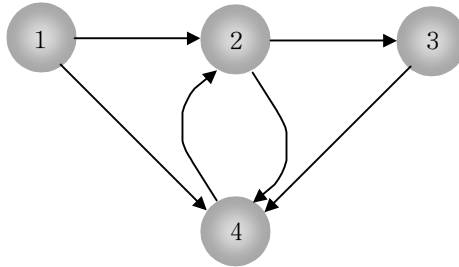



그림 14-5. 프로쎄스와 통로그라프

3. 이것은 분산알고리듬이다. 즉 매개 프로쎄스는 자기자체의 상태와 들어 오는 모든 통로들의 상태를 기록해야 한다.
4. 일단 모든 상태가 기록되였으면(알고리듬이 모든 프로쎄스에서 끝났으면) 알고리듬으로 얻어 진 일관성 있는 전역상태는 매개 프로쎄스가 나가는 모든 통로를 통하여 기록한 상태자료를 송신하게 하고 나가는 모든 통로를 통하여 수신하는 상태자료를 전송함으로써 매개 프로쎄스에서 조립될수 있다. 그대신에 개시하는 프로쎄스는 모든 프로쎄스들을 조사하여 전역상태를 획득할것이다.
5. 알고리듬은 프로쎄스들이 참가하는 임의의 다른 분산알고리듬에 영향을 미치지 않으며 또 그로부터 영향을 받지 않는다.

알고리듬의 사용실례([BEN90]에서 입수한)로서 그림 14-5 에서 설명한 프로쎄스들의 모임을 고찰하자. 매개 프로쎄스는 마디로 표시되며 모든 한 방향통로는 화살표가 가리키는 방향을 가진 두마디사이의 선으로 표시된다. 모든 프로쎄스가 매개의 들어 오는 통로들을 통하여 송신되는 9 개의 통보문을 가지고 순시상기록알고리듬을 실행한다고 가정하자. 프로쎄스 1은 6개 통보문을 송신한후에 전역상태를 기록할것을 결정하며 프로쎄스 4는 3개 통보문을 송신한후에 전역상태를 기록할것을 독립적으로 결정한다. 완료하자마자 모든 프로쎄스로부터 수집한 순시상기록들 즉 결과들을 그림 14-6 에 주었다. 프로쎄스 2 는 상태의 기록보다 먼저 두개의 나가는 통로로 4 개의 통보문을 프로쎄스 3 과 4 에 송신하였다. 그것은 자기의 상태를 기록하기전에 프로쎄스 1 로부터 오는 4 개 통보문을 수신하였으며 통로와 관련된 통보문 5 와 6 을 남겨 둔다. 독자는 일관성을 위하여 순시상기록을 검사하여야 할것이다. 송신된 매개 통보문은 목적프로쎄스에서 수신되였거나 통로에서 이동중에 있는것으로 기록되였다.

분산알고리듬은 강력하고 유연한 도구이다. 그것은 임의의 집중알고리듬을 분산환경에 응용하는데 사용될수 있다. 왜냐하면 임의의 집중알고리듬의 기초는 전역상태에 대한 지식이기때문이다. 특정한 실례들은 교착의 검출과 프로쎄스완료의 검출을 포함한다(실례로 [BENFO], [LYNCF6]을 보시오.). 그것은 또한 고장을 검출하면 분산알고리듬의 검사점을 제공하여 재연산과 회복을 수행하는데 사용될수 있다.

프로쎄스 1 나가는 통로들 2 송신 1,2,3,4,5,6 3 송신 1,2,3,4,5,6 들어 오는 통로들	프로쎄스3 나가는 통로들 2 송신 1,2,3,4,5,6,7,8 들어 오는 통로들 1 수신 1,2,3, 기억 4,5,6 2 수신 1,2,3 기억 4 4 수신 1,2,3
프로쎄스 2 나가는 통로들 3 송신 1,2,3,4 4 송신 1,2,3,4 들어 오는 통로들 1 수신 1,2,3,4,기억 5,6 3 수신 1,2,3,4,5,6,7,8	프로쎄스 4 나가는 통로들 3 송신 1,2,3 들어 오는 통로들 2 수신 1,2 기억 3,4

그림 14-6. 순시상기록의 실례

제 3 절. 분산형호상배제

제 5 장과 제 6 장에서는 기본적으로 병행프로쎄스들의 집행과 관련되는 문제들을 고찰하였다. 제기된 두가지 기본문제는 호상배제와 교착이였다. 제 5 장과 제 6 장은 한개이상의 처리기를 가지지만 하나의 공통주기억기를 가지는 단일체계환경에서 이 문제를 해결하는데 초점을 두었다. 분산조작체계와 공통주기억기 및 박자를 공유하지 않는 처리기들의 집합을 다루는데서는 새로운 난관이 발생하며 새로운 해결방법이 요구된다. 호상배제와 교착을 위한 알고리듬은 통보문교환과 관련되여야 하며 공통기억기에로의 접근에는 관계될수 없다. 이 절과 다음절에서는 분산조작체계환경에서 호상배제와 교착을 고찰한다.

분산호상배제의 개념들

두개이상의 프로쎄스가 체계자원을 사용하려고 경쟁한다면 호상배제를 실시하기 위한 기구가 필요하다. 두개이상의 프로쎄스가 인쇄기와 같은 단일비공유자원에로의 접근을 요구한다고 가정하자. 실행과정에 매개 프로쎄스는 입출력장치에 지령들을 송신하고 상태정보를 수신하며 자료를 송신 및 수신할것이다. 이러한 자원을 림계자원이라고 하며 림계자원과 같은 자원과 그것을 사용하는 프로그람의 부분을 프로그람의 림계구역이라고 한다. 어느 한 순간에 오직 한개의 프로그람만이 자기의 림계구역에 들어 가는것이 중요하다. 상세한 요구가 명백하지 않기때문에 이러한 제약조건을 리해하고 실시하는데서 단순히 조작체계를 믿을수 없다. 실례로 인쇄기의 경우에 임의의 개별적인 프로쎄스가 전체 파일을 인쇄하는 동안 인쇄기에 대한 조종권을 유지하여야 한다. 그렇지 않으면 경쟁하는 프로쎄스들의 선들은 교차될것이다.

프로쎄스들사이의 병행성을 성과적으로 실현하자면 림계구역을 정의하고 호상배제를 실시해야 한다. 이것은 모든 병행처리기구에서 기본이다. 호상배제에 대한 지원을 주는 임의의 기능 또는 능력은 다음의 요구들을 만족시켜야 할것이다.

1. 호상배제를 실시하여야 한다. 즉 어느 한 순간에 같은 자원 또는 공유객체에 대한 림계구역들을 가지는 모든 프로쎄스들중 오직 한개 프로쎄스만이 자기의 림계구

역안에 있는것이 허용된다.
2. 자기의 비림계구역에 정지하는 프로쎄스는 다른 프로쎄스들과 간섭함이 없이 정지하여야 한다.
3. 림계구역에로 접근하는 프로쎄스가 무한정 지체될 가능성이 없어야 한다. 즉 교착 또는 고갈이 전혀 일어 나지 말아야 한다.
4. 림계구역에 들어 와 있는 프로쎄스가 전혀 없으면 림계구역으로 들어 올것을 요구하는 임의의 프로쎄스가 지체없이 들어 오도록 허락되여야 한다.
5. 상대적인 프로쎄스속도 또는 처리기들의 수에 대한 그 어떤 가설도 없다.
6. 프로쎄스는 제한된 시간동안만 자기의 림계구역에 남아 있는다.

그림 14-7 은 분산환경에서 호상배제를 위한 방법들을 시험하기 위하여 사용할수 있는 모형을 보여 준다. 몇가지 형태의 망기구들로 호상 접속된 몇가지 체계를 가정한다. 조작체계의 어떤 함수 또는 프로쎄스가 매개 체계내부에 자원을 배정할수 있다고 가정한다. 이러한 매개 프로쎄스는 몇가지 자원을 조종하며 얼마간의 사용자프로쎄스들을 조종한다. 과제는 이 프로쎄스들이 호상배제를 실시하는데서 협조할수 있는 알고리듬을 작성하는것이다.

호상배제를 위한 알고리듬들은 분산될수도 있고 집중될수도 있다. 완전히 **집중형알고리듬**에서 한개 마디는 조종마디로 지적되여 모든 공유객체들에로의 접근을 조종한다. 임의의 프로쎄스는 림계자원에로의 접근을 요구할 때 자기의 국부자원조종프로쎄스에 요청를 발포한다. 프로쎄스는 공유객체가 사용가능하게 되면 응답(허락)통보문을 되넘기는 조종마디에 요청통보문을 송신한다. 프로쎄스가 자원을 더이상 요구하지 않으면 해제통보문이 조종마디에로 송신된다. 이러한 집중형알고리듬은 두가지 기본특징을 가진다.

1. 오직 조종마디만이 자원배정을 결정한다.
2. 모든 자원들의 신원과 위치 그리고 매개 자원의 배정상태를 포함하는 모든 필요한 정보는 조종마디에 집중된다.

집중방법은 간단하며 호상배제를 실시하는 방법을 리해하기도 쉽다. 조종마디는 자원이 해제될 때까지 자원에 대한 요구를 만족시킬수 없다. 그러나 이러한 기구는 몇가지 약점이 있다. 조종마디가 고장나면 그때는 최소한 림시적으로 호상배제기구가 고장난다. 게다가 모든 자원배정과 재배정은 조종마디와의 통보문교환을 요구한다. 따라서 조종마디는 병목으로 된다.

집중형알고리듬들이 가지고 있는 문제들때문에 분산형알고리듬들의 개발에 더 관심이 돌려 졌다. 완전히 **분산형알고리듬**은 다음과 같은 특성들로 특징 지어 진다[MAEK87]. 즉

1. 모든 마디들은 평균하여 같은 량의 정보를 가진다.
2. 매 마디는 총 체계의 부분적영상만을 가지며 이 정보에 기초하여 결정하여야 한다.
3. 모든 마디들은 마지막결정을 위하여 같은 책임을 지고 있다.
4. 모든 마디들은 마지막결정을 하는데서 평균하여 같은 노력을 소비한다.
5. 일반적으로 마디의 고장은 전체 체계의 붕괴로 귀착되지 않는다.
6. 사건들의 동기를 조종하기 위한 체계규모의 공통박자는 전혀 존재하지 않는다.

2항과 6항은 어느 정도 품이 든다. 2항에 관하여 분산체계들은 임의의 마디에 알려진 모든 정보를 다른 모든 마디들에 전달할것을 요구한다. 이 경우에도 임의의 주어 진 시간에 정보의 일부는 이송중에 있을것이며 다른 모든 마디들에 도착하지 않았을것이다.

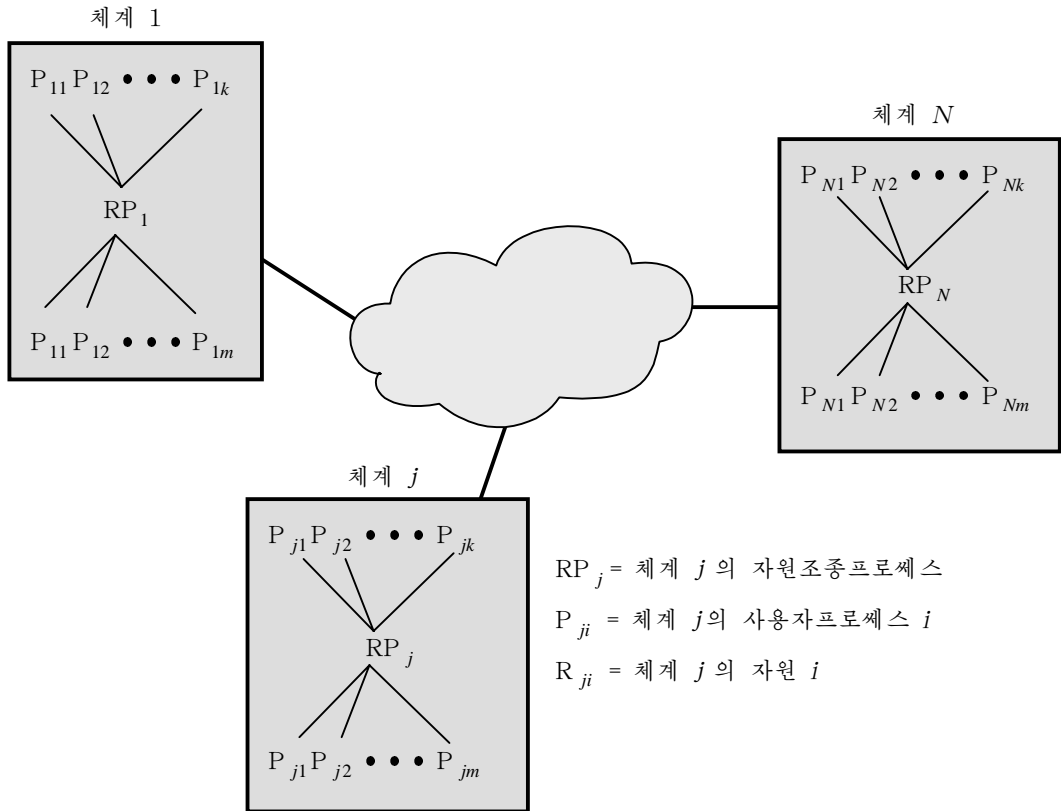



그림 14-7. 분산형프로쎄스관리에서 호상배제문제를 위한 모형

따라서 통보문통신에서의 시간지체때문에 마디의 정보는 보통 완전히 최근의것이 아니며 그런 의미에서 부분적인 정보이다.

6 항에 관하여 체계들사이의 통신에 의한 지연때문에 항상 모든 체계에 유효한 체계규모의 시계를 유지하는것은 불가능하다. 게다가 하나의 중앙시계를 유지하고 중앙시계에 모든 국부시계를 정확히 동기시키는것은 기술적으로 비현실적이다. 어떤 시간주기동안에 동기화의 실패를 일으키는 여러가지 국부시계들사이의 약간의 편차가 있을것이다.

공통시계의 결핍과 결부되는 통신에서의 지연은 집중체계와 견주는 분산체계에서 호상배제기구들의 개발을 더욱더 힘들게 하는 분산호상배제를 위한 몇가지 알고리듬들을 보기전에 시계불일치문제를 극복하기 위한 공통방법을 고찰한다.

분산체계에서 사건의 순서

호상배제와 교착을 위한 대부분의 분산형알고리듬조작에서 기본은 사건의 시간적인 순서이다. 따라서 공통시계 또는 국부시계들을 동기시키는 수단이 없으면 제한을 받는다. 그 문제는 다음과 같은 방법으로 표현될수 있다. 체계 i에서 사건 a는 체계 j에서 사건 b전에(또는 후에) 발생했다고 하자. 그리고 망의 모든 체계에서 이 결론에 일관성 있게 도달한다고 하자. 유감스럽게도 이러한 서술은 두가지 리유로 하여 정확하지 않다. 첫째로, 사건의 실제발생과 어떤 다른 체계에서 그것이 관측된 시간사이에 지연이 있을수 있

다. 둘째로, 동기화의 결핍은 서로 다른 체계의 시계읽기에서 불일치를 가져 온다.

이 난관들을 극복하기 위하여 시간찍기라는 방법이 Lam Port 에 의하여 제안되였다 [LAMP78]. 이 방법은 물리적인 시계를 사용하지 않고 분산체계의 사건들을 배렬한다. 이 수법은 효률적이고 효과적이기때문에 호상배제와 교착을 위한 대부분의 알고리듬들에 사용된다.

먼저 사건이라는 용어를 정의할 필요가 있다. 결국은 프로쎄스가 자기의 림계구역으로 들어 가거나 림계구역을 떠나는것과 같은 국부체계에서 발생하는 동작들에 관심이 있다. 그러나 분산체계에서 프로쎄스들이 대화하는 방법은 통보문들에 관계된다. 그러므로 사건들을 통보문들과 관련시키는것이 적합하다. 국부사건을 통보문에 매우 간단히 결부할수 있다. 실례로 프로쎄스는 자기의 림계구역에 들어 가고 싶을 때 또는 림계구역을 떠나고 싶을 때 통보문을 송신할수 있다. 애매한것을 피하기 위하여 사건을 통보문의 송신과만 관련시키고 통보문의 접수에는 관련시키지 않는다. 따라서 사건은 프로쎄스가 통보문을 전송할 때마다 통보문이 프로쎄스를 떠나는 시간에 대응한다고 정의한다.

시간찍기기구는 통보문들의 전송으로 이루어 지는 사건들을 순서대로 배렬하는데 사용한다. 망에서 매개 체계 i 는 시계로서의 기능을 수행하는 국부계수기 C_i 를 유지한다. 체계가 통보문을 전송할 때마다 국부계수기는 먼저 시계를 하나 증가시킨다. 통보문은 (m, T_i, i)형식으로 송신된다.
여기서

m = 통보문의 내용들
T_i = 통보문을 위한 시간압인. C_i와 같게 설정한다.
i = 싸이트의 수자식별자

통보문을 수신할 때 수신체계 j 는 자기의 현재값과 들어 오는 시간압인의 최대값보다 하나 더 큰 값을 자기 시계에 설정한다.

$$C_j \leftarrow 1 + \max[C_j, T_i]$$

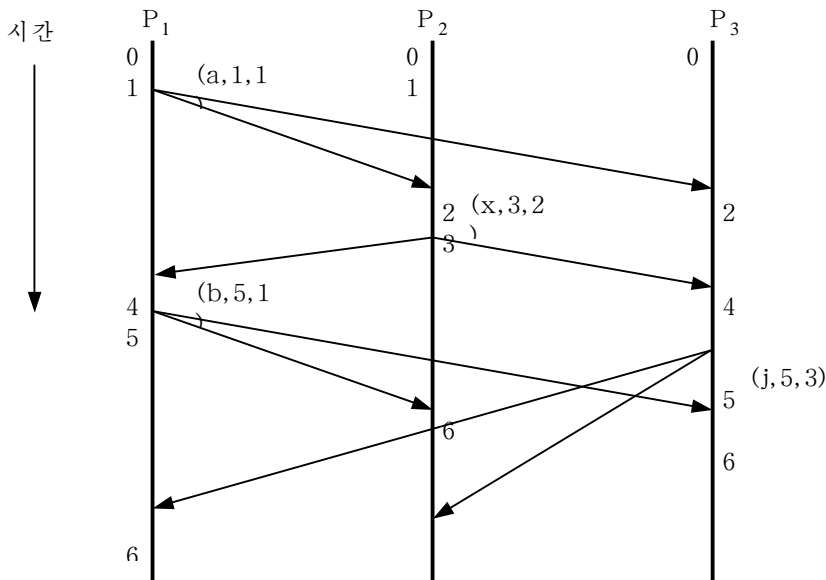



그림 14-8. 시간찍기알고리듬의 조작실례

매개 싸이트에서 사건들의 순서는 다음의 규칙으로 결정된다. 싸이트 i 로부터의 통보문 x와 싸이트 j로부터의 통보문 y에 대하여 다음조건들중 하나가 유지되면 x는 y보다 먼저 일어 난다고 한다.

1. If $T_i < T_j$ or
2. If $T_i = T_j$ and $i < j$

매개 통보문과 관련되는 시간은 통보문을 동반하는 시간압인이며 이 시간들의 순서는 앞에서 말한 두가지 규칙으로 결정된다. 즉 같은 시간압인을 가진 통보문들은 자기들의 싸이트번호에 의하여 순서대로 배렬된다. 이 규칙들의 적용은 싸이트와 관계 없기때문에 이 방법은 통신프로쎄스들의 여러가지 시계들사이의 편차로 인한 어떤 문제도 회피한다.

알고리듬의 조작실례를 그림 14-8 에 주었다. 매개가 시간찍기알고리듬을 조종하는 프로쎄스로 표현되는 세개의 싸이트가 있다. 프로쎄스 P_1 은 시계값 0 에서 시작한다. 통보문 a 를 전송하기 위하여 P_1 은 자기 시계를 하나 증가시키고 (a, 1, 1)을 전송한다. 여기서 첫 수자값은 시간압인이고 두번째 수자값은 싸이트의 식별자이다. 이 통보문을 싸이트2와 3에 있는 프로쎄스들이 수신한다. 두 싸이트의 국부시계는 0의 값을 가지며 2=1 + max[0, 1]의 값으로 설정된다. P_2 는 다음 통보문을 출구하고 먼저 자기의 시계를 3 으로 증가시킨다. 이 통보문을 접수하자마자 P_1 과 P_3 은 자기 시계들을 4 로 증가시켜야 한다. 그다음에 거의 같은 시간에 같은 시간압인을 가지고 P_1 은 통보문 b 를, P_3 은 통보문 j를 출구한다. 순서대로 배렬하는 원리를 사전에 대략 설명하였기때문에 이것은 혼돈을 전혀 가져 오지 않는다. 이 모든 사건들이 일어 난후에 통보문들의 순서는 모든 싸이트들에서 같아 진다. 즉 {a, x, b, j}이다.

알고리듬은 그림 14-9 에서 설명한바와 같이 쌍을 이루는 체계들사이의 전송시간이 차이나도 동작한다. 여기서 P_1 과 P_4 는 같은 시간압인을 가지는 통보문들을 출구한다. P_1 로부터 오는 통보문은 싸이트 2에 P_4의 통보문보다 더 일찍 도착하지만 싸이트 3에서는 P_4 의 통보문보다 더 늦게 도착한다. 그럼에도 불구하고 모든 통보문들이 모든 싸이트에서 수신완료되였으면 통보문의 배렬은 모든 싸이트에서 같다. 즉 {a, q}이다.

이 기구에 의하여 작성된 순서는 실제의 시간순서에 반드시 대응하지 않는다. 시간찍기기구에 기초한 알고리듬에서 어느 사건이 실제로 먼저 일어 났는가는 중요하지 않다. 다만 알고리듬을 실현하는 모든 프로쎄스들이 그 사건들에 부과된 순서를 인정하는것이 중요하다.

방금 론의한 두 실례에서 매개 통보문은 한 프로쎄스로부터 다른 모든 프로쎄스에로 송신된다. 일부 통보문들이 이 방법으로 송신되지 않으면 어떤 싸이트들은 체계의 모든 통보문들을 수신하지 않으며 그러므로 모든 싸이트들이 같은 통보문들의 순서를 가지는 것은 불가능하다. 이러한 경우에 부분적배렬들의 집합이 존재한다. 그러나 우리는 호상배제와 교착검출을 위한 분산형알고리듬에서 시간압인의 사용에 1차적으로 관심을 가진다.

이러한 알고리듬들에서 프로쎄스는 보통 다른 매개 프로쎄스(시간압인을 가진)에 통보문을 송신하며 시간압인은 통보문이 처리되는 방법을 결정하는데 사용된다.

분산대기렬

제 1 판본

분산호상배제를 위하여 제일 먼저 제안된 방법들중의 하나는 분산대기렬의 개념에 기초한것이다[LAMP78]. 알고리듬은 다음과 같은 가정에 기초하고 있다.

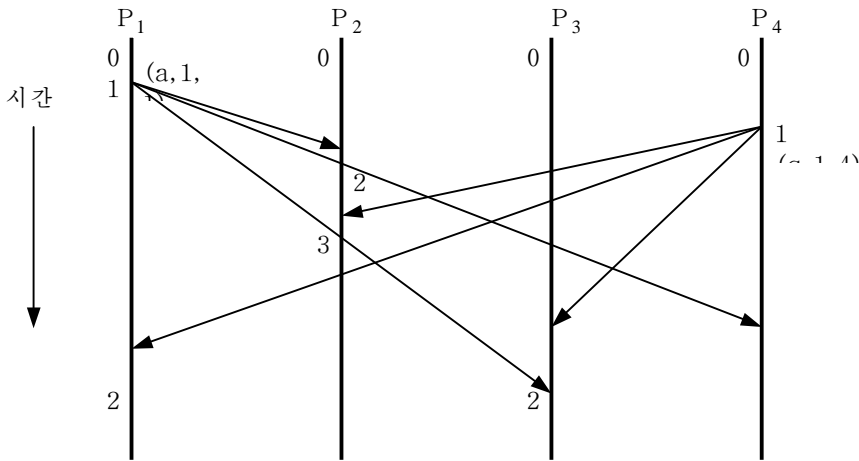



그림 14-9. 시간찍기알고리듬의 또다른 조작실례

1. 분산체계는 유일하게 1 부터 N 까지의 번호가 붙은 N 개의 마디들로 구성되여 있다. 매개 마디는 다른 프로쎄스들을 대표하여 자원들에 대한 서로 배타적인 접근을 요구하는 하나의 프로쎄스를 포함한다. 이 프로쎄스는 역시 동기를 맞추어 중복되여 들어 오는 요청들을 해결하기 위한 중재자로서 봉사한다.
2. 한 프로쎄스에서 다른 프로쎄스에로 송신된 통보문들은 그것들이 송신된것과 같은 순서로 수신된다.
3. 매 통보문은 제한된 시간에 자기 목적지에 정확히 배달된다.
4. 망은 완전히 련결되여 있다. 이것은 매 프로쎄스가 통보문을 전송하기 위하여 중간프로쎄스를 요구하지 않고 다른 매개 프로쎄스들에 직접 통보문들로 송신할수 있다는것을 의미한다.

가정 2와 3은 TCP (부록 1을 보라.)와 같은 믿음직한 전송층규약을 사용하여 실현될수 있다.

간단화를 위하여 매개 싸이트가 오직 단일자원만을 조종하는 경우의 알고리듬을 서술한다. 다중자원들에로의 일반화는 쉽다.

알고리듬으로는 집중체계에서 간단한 방식으로 동작하는 알고리듬을 일반화하려고 한다. 단일중심프로쎄스가 자원을 관리했다면 그것은 들어 오는 요청들을 대기렬에 넣고 선입선출방식으로 요청들을 받아 들였을것이다. 분산체계에서 이와 같은 알고리듬을 달성하기 위하여 모든 싸이트들은 같은 대기렬의 사본을 가져야 한다. 시간찍기는 모든 싸이트들이 모든 요청들을 들어 주는 순서를 공동으로 인정한다는것을 보증하는데 사용된다. 한가지 복잡한 문제가 생긴다. 즉 통보문들이 망을 통과하는데 제한된 시간이 걸리기 때문에 두개의 서로 다른 싸이트는 프로쎄스가 대기렬의 머리부에 있는것을 동의하지 않을것이다. 그림 4-19 를 고찰하자. 통보문 a 가 P_2에 도착했고 통보문 q 가 P_3에 도착했지만 이 두 통보문은 아직 다른 프로쎄스들에로 이송중에 있다. 따라서 P_1 과 P_2 는 통보문 a가 대기렬의 머리부라고 간주하고 P_3과 P_4는 통보문 q가 대기렬의 머리부라고 간주하는 어떤 시기가 있다. 이것은 호상배제의 요구를 위반하게 할것이다. 이것을 피하기 위하여 다음의 규칙을 적용한다. 자기자체의 대기렬에 기초하여 배정을 결정하는 프로쎄스는 자기자체의 대기렬머리부보다 더 먼저 아직 이송중에 있는 통보문은 전혀 없다는것

을 보증하는 다른 모든 싸이트로부터의 통보문을 이미 수신하였어야 한다.

매개 싸이트에서 수신한 가장 최근의 통보문(이 싸이트에서 발생한 가장 최근의 통보문도 포함하여)을 계속 기록하는 자료구조가 매개 싸이트에 있다. Lamport는 이 구조를 대기렬이라고 이름 지었다. 실제로 그것은 한개 입구를 가진 매개 싸이트의 순서이다. 임의의 순간에 국부순서에 있는 입구 q [j]는 P_j로부터 오는 통보문을 포함한다. 순서는 다음과 같이 초기화된다.

$$q[j] = (\text{해방},\ 0,\ j) \quad j = 1, \ldots, N$$

세개의 통보문형태가 알고리듬에서 사용된다. 즉

- (요청, T_i, i) : 자원에 대한 접근은 P_i에서 요구한다.
- (응답, T_j, j): P_j는 자기 조종하에 있는 자원에로의 접근을 승인한다.
- (해방, T_k, k): P_k는 자기에게 이전에 배정된 자원을 해방한다.

알고리듬은 다음과 같다.

1. P_i는 자원에로의 접근을 요구할 때 현재 국부시계값이 찍힌 요청(요청, T_i, i)를 출구한다. P_i는 이 통보문을 자기자체의 대기렬에 있는 q[i]에 넣고 다른 모든 통보문들에 그 통보문을 송신한다.
2. P_i는 (요청, T_i, i)를 수신하면 자기자체의 대기렬에 있는 q[i]에 이 통보문을 넣는다. q[j]가 요청통보문을 포함하지 않으면 그때 P_j는 P_i에로 (응답, T_j, j)를 전송한다. 이것은 결정할 때 이송중에 있는 더 앞선 요구통보문이 더이상 없다는것을 보증하는 이전의 규칙을 실현한다.
3. P_i는 다음의 두 조건이 만족하면 자원에 접근할수 있다(자기 림계구역에 들어 갈수 있다.). 즉
   ㄱ) 순서 q에서 P_i가 소유하는 요청통보문은 순서에서 맨 처음의 요청통보문이다. 통보문들은 모든 싸이트들에 시종일관하게 배렬되기때문에 이 규칙은 임의의 순간에 하나의 프로쎄스만이 자원을 호출하도록 허락한다.
   ㄴ) 국부순서에 있는 모든 통보문들은 q[i]에 있는 통보문보다 더 후에 있다. 이 규칙은 P_i가 자기의 현재 요청보다 먼저 발생한 모든 요청들을 안다는것을 보증한다.
4. P_i는 해방(해방, T_j, j)를 출구하여 자원을 해방하며 자기소유의 순서에 그 통보문을 넣고 다른 모든 프로쎄스들에 전송한다.
5. P_i는 (해방, T_j, j)를 수신할 때 q[j]의 현재 내용들을 이 통보문으로 교체한다.
6. P_i는 (응답, T_j, j)를 수신할 때 q[j]의 현재 내용들을 이 통보문으로 교체한다.

이 알고리듬은 호상배제를 실현하고 공평하며 교착과 고갈을 피한다는것을 쉽게 알수 있다.

- **호상배제** : 림계구역의 입구에 대한 요청들은 시간찍기기구로 작성된 통보문들의 순서에 따라 조정된다. 일단 P_i가 자기의 림계구역에로 들어 가려고 결심하면 자기의 요청통보문전에 전송되는 다른 요청통보문은 체계에 더는 있을수 없다. 이것은 P_i가 그때까지 모든 다른 싸이트들로부터의 통보문들을 반드시 수신했으며 다른 싸이트로부터의 통보문들은 P_i 요청통보문보다 더 늦게 전송하기때문이다. 이것은 응답통보문기구로부터 확신할수 있다. 두 싸이트사이의 통보문들은 순서를 어기고

도착할수 없다.

- **공평 :** 요청들은 시간압인순서에 기초하여 엄격하게 승인된다. 그러므로 모든 프로쎄스들은 같은 기회를 가진다.
- **교착면제 :** 시간압인순서가 모든 싸이트들에 시종일관하게 유지되기때문에 교착은 발생할수 없다.
- **고갈면제 :** P_i는 일단 자기의 림계구역을 완성하면 해방통보문을 전송한다. 이것은 모든 다른 싸이트들에서 P_i의 요청통보문을 삭제하게 하여 다른 프로쎄스가 자기의 림계구역에 들어 가도록 한다.

이 알고리듬이 일정한 효률로 배타를 보증하자면 3×(N-1)개의 통보문이 요구된다. 즉 (N-1)개의 요청통보문, (N-1)개의 응답통보문, (N-1)개의 해방통보문이 요구된다.

제 2 판본

보다 완성된 Lamport 알고리듬은 [RACA81]에서 제안되였다. 그것은 해방통보문들을 제거하는것으로 원래의 알고리듬을 최대로 활용하려고 한다. 한 프로쎄스에서 다른 프로쎄스에로 송신된 통보문들은 그들이 송신된것과 같은 순서로 수신될 필요는 없다는 것을 제외하고 앞에서와 같은 가정이 성립한다.

앞에서와 같이 매개 싸이트는 자원을 조종하는 한개 프로쎄스를 포함한다. 이 프로쎄스는 q를 유지하며 다음의 규칙에 따른다.

1. P_i는 자원에로의 접근을 요구할 때 현재 국부시계값이 새겨 진 요청(요청, T_i, i)를 출구한다. P_i는 이 통보문을 자기의 q[i]에 있는 배렬에 넣고 그 통보문을 모든 다른 프로쎄스들에 송신한다.
2. P_j는 (요청, T_i, i)를 수신할 때 다음의 규칙에 따른다. 즉
   ㄱ) P_j는 현재 자기의 림계구역에 있으면 응답통보문의 송신을 연기한다(규칙 4를 보시오.)
   ㄴ) P_j가 자기의 림계구역에로의 입장을 기다리지 않으면(여전히 주목할만한 요청통보문을 출구하지 않았으면) (응답, T_j, j)를 P_i에로 전송한다.
   ㄷ) P_j가 자기의 림계구역에로의 입장을 기다리고 있으며 그리고 들어 오는 통보문들이 P_j의 요청통보문의 다음에 온다면 자기소유의 q[j]에 있는 배렬에 통보문을 넣고 응답통보문의 송신을 연기한다.
   ㄹ) P_j가 자기림계구역에로의 입장을 기다리고 있고 들어 오는 통보문이 P_j의 요구보다 앞선다면 자기소유의 q[j]에 있는 배렬에 통보문을 넣고 (응답, T_j, j)를 P_i에 전송한다.
3. P_i는 다른 모든 프로쎄스들로부터 응답통보문을 수신하였을 때 자원에 접근할수 있다(자기 림계구역에 들어 갈수 있다.).
4. P_i는 자기 림계구역을 떠날 때 응답통보문을 매개 미결요청통보문에 송신하는것으로 자원을 해방한다.

매개 프로쎄스의 상태이행도를 그림 14-10에 주었다.

요약한다면 프로쎄스는 자기 림계구역에 들어 가고 싶으면 다른 모든 프로쎄스들에 시간압인된 요청통보문을 송신한다. 프로쎄스는 다른 모든 프로쎄스들로부터 응답통보문을 수신할 때 자기 림계구역에 들어 갈수 있다. 프로쎄스는 다른 프로쎄스로부터 요청통보문을 수신한 다음에야 그에 대응한 응답을 송신하여야 한다. 프로쎄스가 자기의 림계

구역에 들어 가고 싶지 않으면 응답통보문을 즉시에 송신한다. 프로쎄스가 자기의 림계구역으로 들어 가고 싶으면 자기 요청통보문의 시간압인을 수신된 마지막요구통보문의 압인과 비교하며 만일 후자가 더 최근의것이라면 자기의 응답통보문을 연기한다. 그밖의 경우에는 응답통보문이 즉시에 송신된다.

이 방식에서는 $2 \times (N\text{-}1)$개의 통보문 즉 자기 림계구역에 들어 가려는 P_i의 의지를 가리키는 $(N\text{-}1)$개의 요청통보문과 요청되는 접근을 허용하는 $(N\text{-}1)$개의 응답통보문이 필요하다.

알고리듬에서는 시간찍기를 사용하여 호상배제를 실현한다. 또한 교착을 회피한다. 후자를 증명하기 위하여 정반대로 즉 이송중에 있는 통보문이 전혀 없을 때 매개 프로쎄스가 요청통보문을 전송하였으며 필요한 응답통보문을 수신하지 못한 상태에 있는것이 가능하다고 가정하자. 이 상태는 발생할수 없다. 왜냐하면 응답통보문을 연기하는 결정은 요청통보문들을 순서대로 배렬하는 관계에 있기때문이다. 그러므로 가장 이른 시간압인을 가지며 모든 필요한 응답들을 수신하는 하나의 요청통보문이 존재한다. 그러므로 교착은 불가능하다.

요청통보문들은 배렬되기때문에 고갈도 역시 피할수 있다. 요청통보문들은 순서배렬로 주어 지기때문에 매개 요청통보문들은 어떤 시기에 가장 오랜 통보문으로 되여 봉사를 받게 된다.

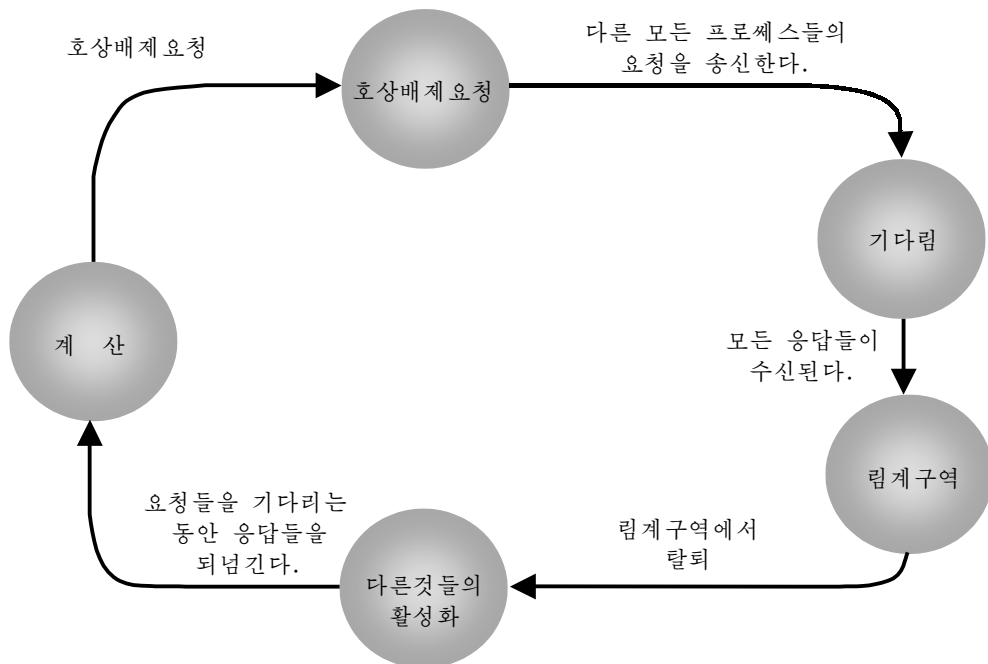



그림 14-10. [RICA81]에서 준 알고리듬의 상태선도

통표넘기기방법

일부 연구자들은 관계하는 프로쎄스들사이에 통표넘기기를 동반하는 완전한 호상배제방법을 제기하였다. 통표는 임의의 시각에 한개 프로쎄스에 유지되는 실체이다. 통표를 유지하고 있는 프로쎄스는 허가를 받지 않고 자기의 림계구역에 들어 갈수 있다. 프로쎄

스는 자기 림계구역에서 나올 때 다른 프로쎄스에 통표를 넘긴다.

이 소절에서는 이 기구들중에서 가장 효과적인 기구를 고찰한다. 그것은 먼저 [SUZU82]에서 제안되였다. 그와 론리적으로 동등한 제안이 [RICA83]에서도 제기되였다. 이 알고리듬에서는 두개의 자료구조가 요구된다. 프로쎄스사이로 넘겨 지는 통표는 실제상 배렬이며 그의 k번째 요소는 통표가 프로쎄스 P_k에 머문 마지막시각의 시간압인을 기록한다. 게다가 매개 프로쎄스는 배렬을 유지하며 그의 j번째 요소는 P_j로부터 수신된 마지막요청통보문의 시간압인을 기록한다.

절차는 다음과 같다. 초기에 통표는 임의로 한개 프로쎄스에 할당된다. 프로쎄스가 자기 림계구역을 사용하고 싶을 때 통표를 가지고 있으면 자기 림계구역을 사용할수 있다. 그렇지 않으면 프로쎄스는 시간압인된 요청통보문을 다른 모든 프로쎄스들에 발송하며 통표를 수신할 때까지 기다린다. 프로쎄스 P_j는 자기의 림계구역에서 나올 때 통표를 다른 프로쎄스에 발송하여야 한다. 프로쎄스 P_j는 P_k의 마지막통표요청에 대한 시간압인이 P_k의 마지막 통표유지를 위하여 통표에 기록된 값보다 더 크기때문에 즉 요청[k] >통표[k]이기때문에 첫번째 입구요청[k]에 대하여 j+1, j+2,⋯,1, 2, ⋯, j-1 순서로 요청배렬을 조사하여 통표를 수신할 다음 프로쎄스를 선택한다.

```
if (!token_present)
{
    clock++;
    broadcast (Request, clock, i);                         서막*/
    wait (access, token);
    token_present = true;
}
token_held = true;
<critical section>;

token[i] = clock;
token_held = false;                                        /*종말부*/
for(intj =i + 1;j <n;j++)
{
   if(request(j) > token [j] && token_present)
   {
     token_present = false;
     send (access, token[j]);
   }
}
for(j= 1;j<=i-1;j++)
{
    if(request(j) > token [j] && token_present)
    {
      token_present = false;
      send(access, token[j]);
     }
```

```
}
                                ㄱ)
if (received (Request, k,j))
{
    request (j) = max(request(j), k);
    if (token_present && ! token_held)
            <text of postlude>;}
}                               ㄴ)
```

Notation

send (j, access, token)	통표를 가진 프로쎄스 j의 접근형송신통보문
broadcast (request, clock, i)	다른 모든 프로쎄스에로의 프로쎄스의 시간압인 clock를 가진 요청형송신통보문
received (request, t, j)	시간압인 t를 가진 프로쎄스 j의 요청형수신통보문

그림 14-11. (프로쎄스 P_i에 대한) 통표넘기기알고리듬: ㄱ-첫 부분, ㄴ-두번째 부분

그림 14-11에서는 두개 부분으로 된 알고리듬을 보여 주고 있다. 첫 부분은 림계구역의 사용을 취급하며 서막과 림계구역, 종말부로 구성되여 있다. 두번째 부분은 요청의 접수를 담당하는 작용과 관련된다.

변수 clock 는 시간압인기능에 사용되는 국부계수기이다. 조작 wait(접근, 통표)는 《접근》형통보문을 수신할 때까지 프로쎄스를 기다리게 한다. 접근형통보문을 수신하면 그것을 변수배렬통표안에 넣는다.

알고리듬은 다음 두가지 요구상태중 어느 한 상태에 있을수 있다.

- 요청하는 프로쎄스가 통표를 가지고 있지 않을 때 *N* 개 통보문(요청를 발송하기 위하여 *N*-1 개, 통보를 이동시키기 위하여 1)을 요구한다.
- 프로쎄스가 이미 통표를 가지고 있다면 통보문이 전혀 요구되지 않는다.

제 4 절. 분산형교착

제 6 장에서는 체계자원에 대하여 경쟁하거나 서로 통신하는 프로쎄스들의 모임의 영구적인 봉쇄로서 교착을 정의하였다. 이 정의는 분산체계에서는 물론 단일체계에서도 유효하다. 호상배제의 경우와 마찬가지로 교착은 공유기억기체계와 비교되는 분산체계에서 더 복잡한 문제들을 제기한다. 마디들이 전반 체계의 현재상태에 대한 정확한 지식을 전혀 가지고 있지 않기때문에 그리고 프로쎄스들사이의 매개 통보문이송은 예측할수 없는 지연을 동반하기때문에 분산체계에서의 교착조정은 복잡해 진다.

두가지형의 분산교착이 문헌에서 주목을 끌었다. 즉 자원들의 배정에서 발생하는 분산교착과 통보문들의 통신으로 일어 나는 분산교착이다. 자원교착들에서 프로쎄스들은 자료기지에서 자료객체들 또는 봉사기에 있는 입출력자원들과 같은 자원들에 접근하려고 한다. 프로그람의 모임안에 있는 매개 프로쎄스가 다른 프로쎄스가 유지하는 자원을 요구하면 교착이 발생한다. 통신교착들에서 통보문들은 프로쎄스들이 기다리는 자원들이다. 교착은 모임에 있는 매개 프로쎄스가 그 모임에 있는 다른 프로쎄스로부터의 통보문을 기다리고 그 모임에 있는 그 어떤 프로쎄스가 통보문을 송신하지 않을 때 발생한다.

자원배정에서 교착

제 6 장으로부터 자원배정에서의 교착은 다음의 조건들이 모두 성립할 때에만 존재한다는것을 알수 있다.

- **호상배제 :** 어느 한 순간에 오직 한개 프로쎄스만이 어떤 자원을 사용할수 있다.
- **유지 및 기다림 :** 프로쎄스는 다른 자원들의 할당을 기다리면서 배정된 자원들을 유지할수 있다.
- **비선취 :** 그 어떤 자원도 자기를 유지하는 프로쎄스로부터 강제로 완전히 제거될수 없다.
- **순환기다림 :** 사슬의 매개 프로쎄스는 사슬의 다음번 프로쎄스가 요구하는 최소한 하나의 자원을 유지하는것과 같은 프로쎄스들의 닫긴 사슬을 가진다.

교착을 처리하는 알고리듬의 목적은 순환기다림형성을 방지하거나 그의 실제적인 발생 또는 잠재적인 발생을 검출하는것이다. 게다가 자원들은 많은 싸이트들에 분산되며 그 자원들에로의 접근은 체계전역상태에 대한 완성된 최신정보를 가지고 있지 않는 조종프로쎄스들에 의하여 조절된다. 그러므로 프로쎄스들은 국부정보에 기초하여 결정하여야 한다. 따라서 새로운 교착알고리듬들이 요구된다.

분산교착관리에서 제기되는 난관의 한가지 실례는 유령교착현상이다. 유령교착의 실례를 그림 14-12 에서 설명한다. 표시법 $P_1 \rightarrow P_2 \rightarrow P_3$은 P_1이 P_2가 유지하는 자원을 기다려 정지되고 P_2가 P_3이 유지하는 자원을 기다려 정지된다는것을 의미한다. 초기에 P_3은 자원 Ra 를 소유하고 P_1은 자원 Rb 를 소유한다고 하자. 이제 P_3이 Ra 를 해제하는 통보문을 먼저 출구하고 그다음에 Rb를 요구하는 통보문을 출구한다고 가정하자. 첫 통보문이 두번째 통보문보다 먼저 주기검출프로쎄스에 도달하면 자원요구항목들을 적당히 반영하는 그림 14-12 ㄱ의 결과가 발생한다.

그러나 두번째 통보문이 첫번째 통보문보다 먼저 도착하면 교착이 기록된다(그림 14-12 ㄴ). 이것은 집중체계에서 있을수 있는것과 같은 전역상태의 결핍으로 인한 진짜 교착이 아니며 잘못된 검출이다.

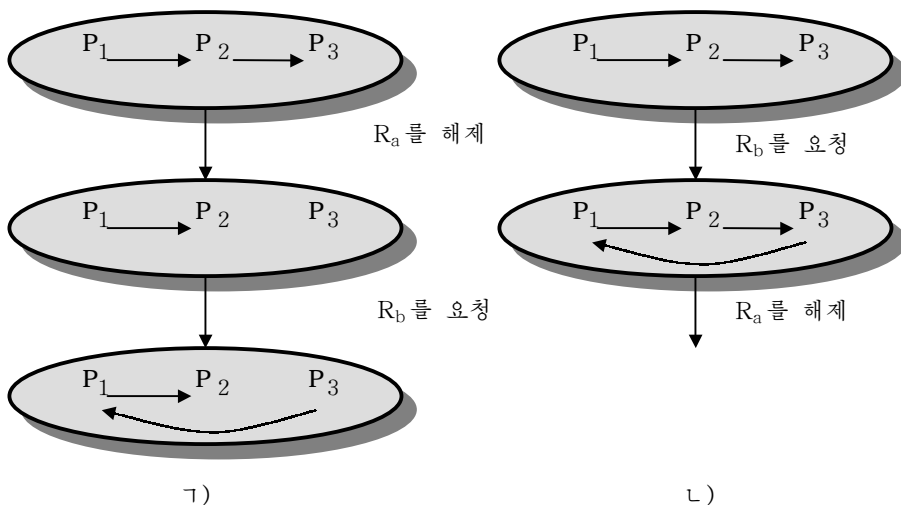



그림 14-12. 유령교착
ㄱ-해제통보문은 요청통보문전에 도착한다, ㄴ-요청통보문은 해제통보문전에 도착한다.

교착방지

제 6 장에서 론의한 교착예방수법중 두가지는 분산체계에서 사용할수 있다.

1. 자원형태들의 선형순서를 정의하여 순환기다림조건을 예방할수 있다. 프로쎄스는 R 형태의 자원들을 배정받으면 순서에서 R 다음에 오는 형태의 자원들만을 요청한다. 이 방식의 기본결함은 자원들이 사용되는 순서로 자원들을 요청할수 없다는것 따라서 자원들은 필요한 시간보다 더 오래 유지될수 있다는것이다.
2. 프로쎄스가 한꺼번에 필요한 모든 자원들을 요청하도록 규정하는것으로 또한 모든 요청들을 동시에 허락할수 있을 때까지 프로쎄스를 봉쇄하는것으로 유지 및 기다림조건을 예방할수 있다. 이 방법은 두가지 점에서 비효률적이다. 첫째로, 어떤 프로쎄스는 사실 몇개 자원만으로 수행될수 있었지만 자기가 요청하는 모든 자원이 준비될 때까지 오랜 시간동안 유지될수 있다. 둘째로, 어떤 프로쎄스에 배정된 자원들은 다른 프로쎄스들에 배정되지 않는 상당한 기간 사용되지 않은 채로 있을수 있다.

이 두 방식은 프로쎄스가 사전에 자기의 지원요청들을 결정할것을 요구한다. 사실은 항상 그렇지 않다. 실례로 새로운 항목을 동적으로 추가할수 있는 자료기지응용을 들수 있다. 이러한 예지를 요구하지 않는 방법의 실례로서 [RUSE78]에서 제안된 두개 알고리듬을 고찰한다. 이것들은 자료기지작업환경에서 개발되였기때문에 프로쎄스라고 하지 않고 트랜잭션이라고 한다.

제안된 방법들은 시간압인을 사용한다. 매개 트랜잭션은 자기의 생존기간동안에 생성시간압인을 진행한다. 이것은 트랜잭션에서 엄격한 순서를 확립한다. 트랜잭션 T1 이 이미 사용하고 있는 자원 R 는 다른 트랜잭션 T2 가 요구하면 자기들의 시간압인을 비교하여 충돌을 해결한다. 이 비교는 순환기다림조건의 성립을 예방하는데 사용된다. 이 기초방법의 두가지 변종 즉《기다림-죽음》방법과《부상-기다림》방법이 저자들에 의하여 제안되였다.

T1 은 현재 R 를 유지하며 T2 는 요청을 출구했다고 가정하자. 그림 14-13 ㄱ는 **기다림-죽음방법**을 위하여 자원배정자가 R 의 싸이트에서 사용하는 알고리듬을 보여 준다. 두개의 트랜잭션의 시간압인을 e(T1)과 e(T2)로서 표시한다. T2이 더 오랜것이면 그것은 해제통보문을 능동적으로 출구하든가 다른 자원을 요청할 때 죽든가 하는것으로 T1 이 R를 해제할 때까지 봉쇄된다. T2이 더 새로운것이지만 그것은 전과 같은 시간압인을 가지고 재시동된다.

```
If (e(T2)<e(Tl))
halt_T2 ('wait');
else
kill_T2 ('die');
ㄱ)
```

```
if (e(T2)<e(Tl))
kill_Tl ('wound');
else
halt_T2 ('wait');
ㄴ)
```

그림 14-13. 교착예방방법들

ㄱ-기다림-죽음방법, ㄴ-부상-기다림방법

따라서 충돌에서는 오랜 트랜잭션이 우선권을 가진다. 죽은 트랜잭션은 자기의 원래 시간압인을 가지고 다시 살아나기때문에 그것은 더 오랜것으로 되며 따라서 증가된 우선권을 획득한다. 모든 자원들의 배정상태를 알려고 하는 싸이트는 하나도 없다. 모두 요구

되는것은 자기자원들을 요구하는 트랜잭션들의 시간압인들이다.

부상-기다림방법은 요구된 자원을 사용하고 있는 더 새로운 트랜잭션을 죽이고 오랜 트랜잭션의 요구를 즉시에 허락한다. 이것을 그림 14-13 ㄴ에서 보여 준다. 기다림-죽음방법과는 달리 어떤 트랜잭션은 자기보다 새로운 트랜잭션이 사용하고 있는 자원을 결코 기다리지 말아야 한다.

교착회피

교착회피는 주어 진 자원배정요구를 할수 있는 경우 교착을 일으키겠는가를 동적으로 결정하는 수법이다. [SING946]는 분산교착회피가 다음과 같은 리유로 하여 비현실적이라고 지적한다.

1. 매개 마디는 체계의 전역상태를 추적하여야 한다. 이것은 충분한 기억기의 통신내부조작을 요구한다.
2. 안전한 전역상태를 검사하는 프로쎄스는 서로 배타적이여야 한다. 그렇지 않으면 두마디는 각각 다른 프로쎄스의 자원요구를 고려할수 있으며 요구를 존중하는것이 안전하다는 결론을 동시에 내릴것이다. 사실 두 요구가 다 존중되면 교착이 발생한다.
3. 안전상태를 검사하는것은 많은 프로쎄스의 자원을 가지는 분산체계에 상당한 내부조작처리를 요구한다.

교착검출

교착검출에서 프로쎄스들은 자기들이 요구에 따라 자유로운 자원들을 얻게 되며 교착의 존재는 그후에 결정된다. 교착이 검출되면 교착성분을 이루는 프로쎄스중 한개 프로쎄스가 선택되며 교착을 해소하는데 필요한 자원들을 해제할것이 요구된다.

분산교착검출에서 난관은 교착이 분산자원들을 포함할수 있지만 매개 싸이트는 오직 자기소유의 자원들에 대하여서만 알고 있는것이다. 체계조종이 집중되였는지, 계층화되였는지 또는 분산되였는지에 관계되는 여러가지 방법들이 가능하다(표 14-1).

집중조종의 경우에 한개의 싸이트가 교착을 검출한다. 모든 요청 및 해제통보문들은 특정자원을 조정하는 프로쎄스는 물론 중앙에도 송신된다. 중앙은 완전한 영상을 가지고 있기때문에 교착을 검출할수 있다. 이 방법은 많은 통보문들을 요구하며 중앙싸이트의 고장에 민감하다. 게다가 유령교착들이 검출될수 있다.

계층조종의 경우에 싸이트들은 나무뿌리로 봉사하는 한개 싸이트를 가지는 나무구조로 구성된다. 잎마디들이 아닌 매 마디에서는 모든 종속마디들의 자원배정에 대한 정보를 수집한다. 이로하여 교착검출은 뿌리마디보다 더 낮은 수준들에서 진행된다. 특히 자원들의 모임을 포함하는 교착은 객체들사이에서 충돌하고 있는 자원을 가지는 모든 싸이트들의 공동조상인 마디에 의하여 검출된다.

분산조종의 경우에 모든 프로쎄스들은 교착검출기능에 협력한다. 일반적으로 이것은 시간압인들을 가진 주목할만한 정보를 교환하여야 한다는것을 의미하며 따라서 내부처리가 중요하다. [RAYN88]은 분산조종에 기초한 몇가지 방법들을 주며 [DATT90]는 한가지 방법을 상세히 고찰한다.

분산교착검출알고리듬의 실례를 보자([DATT92], [JOHN91]). 알고리듬은 매개 싸이트가 자료기지의 한 부분을 유지하는 분산자료지지체계를 대상으로 하여 트랜잭션들을 매개 싸이트에서 개시될수 있다. 트랜잭션은 기껏해서 하나의 미해결자원요청를 가질수 있다. 트랜잭션이 한개이상의 자료객체를 요구한다면 두번째 자료객체는 오직 첫번째 자

료객체가 허가된 다음에 요구될수 있다.

싸이트에서 매개 자료객체 i 와 관련되는것은 두개 파라메터 즉 유일식별자 D_i 와 변수 Locked-by(D_i)이다. 후자의 변수는 자료객체가 임의의 트랜잭션에 의하여 페쇄되지 않으면 값 령을 가지며 그렇지 않으면 그 값은 페쇄트랜잭션의 식별자이다.

싸이트에서 매 트랜잭션 j와 관련되는것은 4개 파라메터이다.

- 유일식별자 T_j
- 변수 Held-by(T_j), 트랜잭션 T_j 가 실행중이거나 준비상태이면 령으로 설정된다. 그렇지 않으면 그의 값은 트랜잭션 T_j 가 요구한 자료객체를 유지하고 있는 트랜잭션이다.
- 변수 Wait-for(T_j), 트랜잭션 T_i 가 임의의 다른 트랜잭션을 대기하고 있지 않으면 령값을 가진다. 그렇지 않으면 그의 값은 페쇄된 트랜잭션의 순서목록머리부에 있는 트랜잭션의 식별자이다.
- 대기렬 Request-Q(T_j), T_j가 유지하고 있는 자료객체에 대한 모든 미해결요청들을 포함한다. 대기렬의 매개 요소는 (T_k , D_k)형식으로 되여 있다. 여기서 T_k는 요구트랜잭션이며 D_k는 T_j가 유지하는 자료객체이다.

표 14-1. 분산교착검출전략

집중형알고리듬		계층형알고리듬		분산형알고리듬	
우점	약점	우점	약점	우점	약점
· 알고리듬은 개념적으로 단순하고 실현하기 쉽다. · 중앙싸이트는 완전한 정보를 가지며 교착들을 최적으로 해결할수 있다.	· 통신내 조작이 많다. 매개 마디는 중심마디에 상태정보를 송신하여야 한다. · 중심마디의 고장에 민감하다.	· 단일고장문제의 영향을 받지 않는다. · 대부분의 가능한 교착들이 상대적으로 국한되였으면 교착해결동작이 제한된다.	· 대부분의 가능한 교착들이 국한되도록 체계를 구성하기가 힘들수 있다. · 그밖의 경우에 분산방법에서 더 많은 내부조작이 실제로 있을수 있다.	· 단일고장문제의 영향을 받지 않는다. · 교착검출동작으로 교체되는 마디는 전혀 없다.	· 여러개의 싸이트들이 같은 교착을 검출할수 있으며 교착에 포함된 마디들을 알수 없으므로 교착해결이 부담으로 된다. · 고려할 항목들의 동기때문에 알고리듬을 실현하기 힘들다.

실례로 트랜잭션 T_2 는 T_1 이 유지하는 자료객체를 기다리며 다음에는 T_0 이 유지하는 자료객체를 기다린다고 가정하자. 이때 관련되는 파라메터들은 다음과 같은 값을 가진다. 즉

트랜잭션	Wait_for	Held_by	Request_Q
T_0	nil	nil	T_1
T_1	T_0	T_0	T_2
T_2	T_0	T_1	nil

실례는 Wait-for(T_i)와 Held-by(T_i)의 차이를 강조한다. 어느 프로쎄스도 T_0이 T_1이 요구하고 있는 자료객체를 해제할 때까지 처리를 계속할수 없다.

그림 14-14 는 교착검출에 사용되는 알고리듬을 보여 준다. 트랜잭션이 자료객체에

폐쇄요청를 할 때 자료객체와 관련된 봉사기프로쎄스는 그 요청를 받아 들이든가 또는 거절한다. 요청이 허락되지 않으면 봉사기프로쎄스는 자료객체를 유지하고 있는 트랜잭션의 신원을 자료객체에 대한 폐쇄요청을 한 트랜잭션에 넘겨 준다.

요청트랜잭션이 허락응답을 수신하면 자료객체를 폐쇄한다. 만약 그렇지 않으면 요청트랜잭션은 Held-by 변수를 자료객체를 유지하고 있는 트랜잭션의 신원으로 갱신한다.

요청트랜잭션은 유지트랜잭션의 Request_Q 에 자기 신원을 더한다. 요청트랜잭션은 Wait-for 변수를 유지트랜잭션의 신원(트랜잭션이 기다리고 있지 않으면)이나 유지트랜잭션의 Wait –for 변수의 신원으로 갱신한다. 이런 방법으로 Wait-for 변수는 결국 실행을 페색하고 있는 트랜잭션의 값으로 설정한다. 결국 요청트랜잭션은 자기의 Request –Q 에 있는 모든 트랜잭션들에 갱신통보문을 출구하여 이 변화의 영향을 받는 모든 Wait-for 변수를 변경시킨다.

```
Π/* lock_request(Ti)를 수신하는 자료객체 Dj */
if(Locked_by(Dj) == null)
    send(granted);
else
{
    send not granted to Ti;
    send Locked_by(Dp to Ti)
}

/* 트랜잭션 Ti는 자료객체 Dj에 대한 폐쇄요청을 한다. */
send lock_request(Ti) to Dj;
wait for granted/not granted;
if (granted)
{
    Locked_by(Dj) = Ti;
    Held_by(Ti) = <ˆ;
}
else /* Dj는 트랜잭션 Tj에 의하여 사용되고 있다. */
{
    Held_by(Ti)=Tj;
    Enqueue(Ti, Request_Q(Tj));
    if(Wait_for(Tj)==null)
        Wait_for(Ti) = Tj ;
    else
        Wait_for(Ti) = Wait_for(Tj);
    update(Wait_for(Ti),Request_Q(Ti));
}
/* 갱신통보문을 수신하는 트랜잭션 Tj*/

    if (Wait_for(Tj) != Wait_for(Ti))
        Wait_for(Tj) = Wait_for(Ti);
```

```
if (intersect(Wait_for(Tj), Request_Q(Tj)) = null)
    update(Wait_for(Ti),Request_Q(Tj);
else
{
        DECLARE DEADLOCK.;
        /* 다음과 같이 교착해결을 개시한다.*/
        /* 탈퇴하는 트랜잭션으로 Tj가 선택된다. */
        /* Tj는 자기가 유지하는 모든 자료객체들을 해제한다. */
        send_clear(Tj, Held_by(Tj));
        allocate each data object DI held by Tj to the first
           requester T^ in Request_Q(Tj);
        for (every transaction Tn in Request_Q(Tj) requesting
           data object DI held by Tj)
        {
           Enqueue(Tn, Request_Q(Tk));
        }
}
/* clear (Tj, Tk)통보문을 수신하는 트랜잭션 Tk */
   purge the tuple having Tj as the requesting transaction from
   Request_Q(Tk);
```

그림 14-14. 분산교착검출알고리듬

트랜잭션은 갱신통보문을 수신하면 자기의 Wait-for 변수를 갱신하여 자기가 마지막으로 요청하였던 트랜잭션이 다른 트랜잭션에 의하여 아직 페색되여 있다는 사실을 반영한다. 그다음에 트랜잭션은 자기를 기다리고 있는 프로쎄스들중의 한개 프로쎄스를 자기가 기다리고 있는지를 검사하는것으로 교착검출의 실제작업을 진행한다. 만일 기다리고 있지 않다면 트랜잭션은 갱신통보문을 전송한다. 기다리고 있다면 트랜잭션은 자기가 요청한 자료객체를 유지하고 있는 트랜잭션에 삭제통보문을 송신하고 자기가 유지하고 있는 매개 자료객체를 자기의 Request -Q 에 있는 첫 요청자에게 배정하며 나머지 요청자들을 새로운 트랜잭션의 대기렬에 넣는다.

알고리듬의 조작실례를 그림 14-15 에 주었다. T_0 이 T_3 이 유지하는 자료객체를 요구할 때 주기가 생성된다. T_0 은 T_1 에서 T_2 , T_3 으로 전달되는 갱신통보문을 출구한다. 이 시험에서 T_3 은 자기의 Wait-for 와 Request -Q 변수의 공통부분이 비여 있지 않다는것을 발견한다. T_3 은 Request -Q(T_2)에서 T_3 을 제거하도록 지우기통보문을 송신하며 자기가 유지하고 있는 자료객체를 해제하고 T_4 와 T_6 을 가동시킨다.

통보문통신에서 교착

호상기다림

교착은 통보문통신에서 프로쎄스그룹의 매개 성원이 그룹의 다른 성원으로부터 오는 통보문을 기다리고 있으며 이동중에 있는 통보문이 전혀 없을 때 발생한다.

이 상태를 더 상세히 분석하기 위하여 프로쎄스의 의존모임 (DS)를 정의한다. 통보문을 기다려 정지된 프로쎄스P_i에서 DS(P_i)는 P_i 가 기대하는 통보문을 보내는 모든 프로쎄스들로 구성되여 있다. 대표적으로 P_i 는 기대하던 통보문중 한개가 도착하면 계속 가동할수 있다. 다른 방법은 P_i 가 오직 기대했던 통보문이 모두 도착한후에만 실행할수 있다. 전자의 상태가 더 일반적인 상태이며 여기서 고찰한다.

앞의 정의를 사용하여 프로쎄스들의 모임 S에서의 교착을 다음과 같이 정의할수 있다.

1. 모임 S에 있는 모든 프로쎄스들은 통보문을 기다려 정지된다.
2. S는 S에 있는 모든 프로쎄스들의 의존모임을 포함한다.
3. S의 성원들사이에서 이동중인 통보문은 전혀 없다.

S 에 있는 임의의 프로쎄스는 자기를 해제할 통보문을 결코 수신할수 없기 때문에 교착된다.

그라프적으로 볼 때 통보문교착과 자원교착은 차이난다. 자원교착의 경우에 프로쎄스의존성을 묘사하는 그라프에 닫긴 고리 또는 닫긴 순환이 있으면 교착이 존재한다. 자원교착의 경우 한 프로쎄스는 다른 프로쎄스에 관계되는데 그것은 후자가 전자가 요구하는 자원을 유지할 때이다. 통보문교착의 경우에 교착조건은 S의 임의의 성원의 모든 후계자들이 S에 있는 그들자신이라는것이다. 그림 14-16은 그 문제를 설명한다. 그림 14-16 ㄱ에서 P_1은 P_2 나 P_5에서 오는 통보문을 기다린다. P_5는 임의의 통보문도 기다리지 않으며 따라서 P_1에로 통보문을 송신할수 있으며 이로부터 P_1은 해제된다. 결과(P_1, P_5)와 (P_1, P_2)는 없어 진다. 그림 14-16 은 의존성을 첨부한다. P_5는 P_2에서 오는 통보문, P_2는 P_3에서 오는 통보문, P_3은 P_1에서 오는 통보문, P_1은 P_2에서 오는 통보문을 기다린다. 따라서 교착이 존재한다.

자원교착의 경우와 같이 예방 또는 검출로 통보문교착을 제거할수 있다. [RAYN88]은 몇가지 실례를 제시한다.

통보문완충기들의 비사용률

통보문넘기기체계에서 교착이 발생할수 있는 또다른 상태는 이동중에 있는 통보문들을 기억하기 위한 완충기들을 배정하는 문제와 관계된다. 이러한 종류의 교착은 파케트교환자료망들에서 흔히 볼수 있다. 이 문제를 먼저 자료망환경에서 고찰한다음 분산조작체계의 관점에서 본다.

자료망에서 교착의 가장 단순한 형태는 직접통보절환교착이며 파케트교환마디가 요청중인 파케트를 완충기에 할당하는 공통완충기조합을 사용하면 발생할수 있다. 그림 14-17 ㄱ는 마디 A에 있는 모든 완충기공간을 B로 가는 파케트가 차지한 상태를 보여준다. 어느쪽 마디도 자기 완충기가 다 충만되여 있기때문에 그이상 파케트를 접수할수 없다. 따라서 어느쪽 마디도 임의의 련결에서 송신 또는 수신할수 없다.
모든 완충기들을 단일련결에 전용시키지 않으면 직접통보절환교착을 예방할수 있다. 마디련결에 하나씩 개별적인 고정크기완충기들을 사용하면 이러한 예방을 달성할수 있다. 비록 공통완충기조합을 사용한다고 해도 단일련결이 모든 완충기공간을 전혀 획득할수없으면 교착을 회피한다.

더 교모한 형태의 교착인 간접통보절환교착은 그림 14-17 ㄴ에서 설명한다. 매개 마디에서 한 방향으로 향한 린접마디에로의 대기렬은 그밖에 다른 마디에로 향한 파케트들로 준다.

Transaction	Wait_for	Held_by	Request_Q
T_0	nil	nil	T_1
T_1	T_0	T_0	T_2
T_2	T_0	T_1	T_3
T_3	T_0	T_2	T_4, T_6
T_4	T_0	T_3	T_5
T_5	T_0	T_4	nil
T_6	T_0	T_3	nil

ㄱ)

Transaction	Wait_for	Held_by	Request_Q
T_0	T_0	T_3	T_1
T_1	T_0	T_0	T_2
T_2	T_0	T_1	T_3
T_3	T_0	T_2	T_4, T_6, T_0
T_4	T_0	T_3	T_5
T_5	T_0	T_4	nil
T_6	T_0	T_3	nil

ㄴ)

그림 14—15. 그림 14-14에 준 분산교착알고리듬의 실례

ㄱ-요청전의 체계상태, ㄴ- T_0 이 T_3 에 요청한후의 체계상태

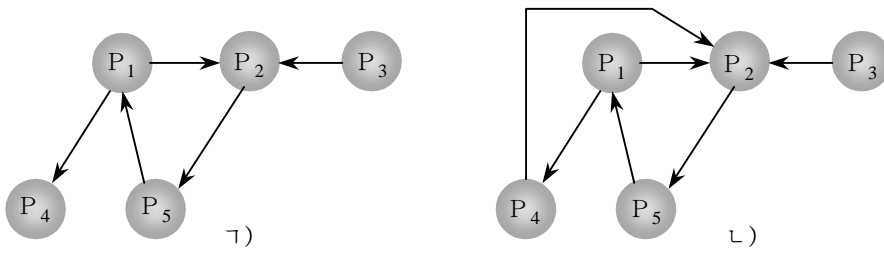



그림 14-16. 통보문에서 교착
ㄱ-비교착, ㄴ-교착

어느쪽 마디도 자기 완충기가 다 충만되여 있기때문에 그이상 파케트를 접수할수 없다. 따라서 어느쪽 마디도 임의의 련결에서 송신 또는 수신할수 없다.

모든 완충기들을 단일련결에 전용시키지 않으면 직접통보절환교착을 예방할수 있다.

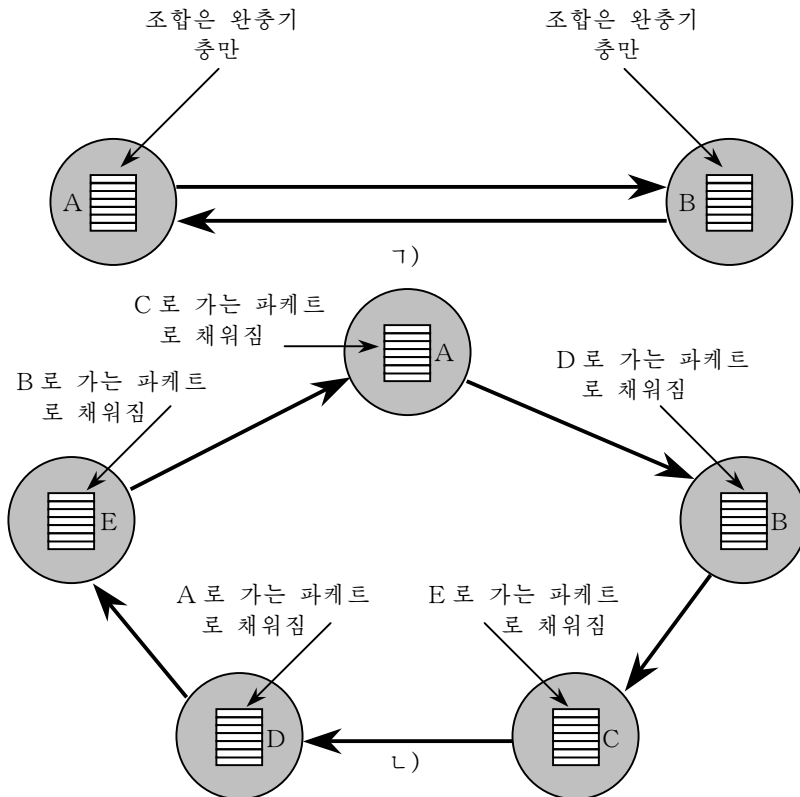



그림 14-17. 통보절환교착
ㄱ-직접통보절환교착, ㄴ-간접통보절환교착

마디련결에 하나씩 개별적인 고정크기완충기들을 사용하면 이러한 예방을 달성할수 있다. 비록 공통완충기조합을 사용한다고 해도 단일련결이 모든 완충기공간을 전혀 획득할수 없으면 교착을 회피한다.

더 교모한 형태의 교착인 간접통보절환교착은 그림 14-17 ㄴ에서 설명한다. 매개 마디에서 한 방향으로 향한 린접마디에로의 대기렬은 그밖에 다른 마디에로 향한 파케트들로 채워 진다. 이런 형태의 교착을 예방하는 한가지 단순한 방법은 구조화된 완충기조합을 채용하는것이다(그림 14-18). 완충기들은 계층형으로 구성된다. 0수준에 있는 기억기조합은 제한이 없으며 임의로 들어 오는 파케트는 거기에 기억될수 있다. 1 수준에서 N 수준(N는 임의의 망경로에 있는 최대홉수이다.)까지 다음의 방법으로 완충기들이 예약된다. 즉 k수준에 있는 완충기들은 지금까지 적어도 k홉을 이동한 파케트들을 위하여 예약된다. 따라서 과부하상태에서 완충기들이 채워 지면 k 또는 그보다 적은 홉을 한 파케트들의 도착은 버려 진다. 이 전략이 직접 및 간접통보절환교착을 제거한다는것을 [GOPA 85]에서 알수 있다.

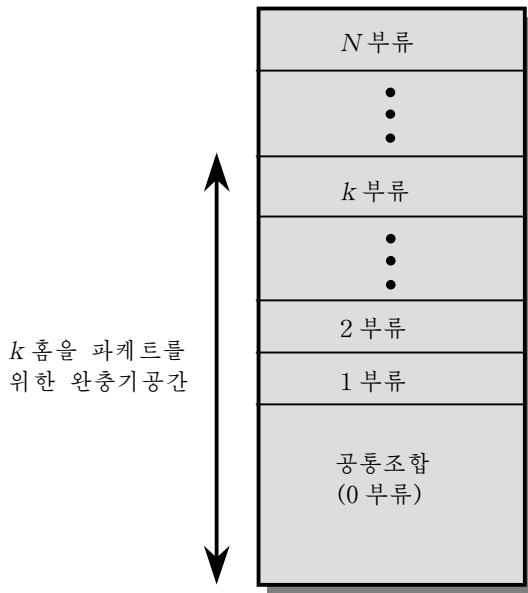



그림 14-18. 교착예방을 위한 구조화된 완충기조합

방금 서술한 교착문제는 통신방식환경에서 대표적으로는 OSI3층(망층)에서 취급될것이다. 같은 부류의 문제는 프로쎄스간통신을 위한 통보문넘기기를 사용하는 분산조작체계에서 발생할수 있다. 특히 송신조작이 비페색이면 나가는 통보문들을 유지하기 위하여 완충기가 요구된다. 프로쎄스 X 에서 프로쎄스 Y 에로 송신되는 통보문들을 유지하기 위하여 사용되는 완충기를 X 와 Y 사이의 통신통로라고 간주할수 있다. 이 통로가 제한된 용량(제한된 완충기크기)을 가진다면 송신조작이 프로쎄스정지로 귀착될수 있다. 즉 완충기의 크기가 n 이고 현재 n 개 통보문이 이동중(아직 목적지프로쎄스에 수신되지 않음)이라면 추가적인 송신의 실행은 수신이 완충기공간을 개방할 때까지 송신하는 프로쎄스를 페색할것이다.

그림 14-19는 제한된 통로들의 사용이 어떻게 교착을 일으킬수 있는가를 설명한다. 그림은 각각 4개 통보문크기를 가지는 두개 통로를 보여 준다. 하나는 프로쎄스 X 에서 프로쎄스 Y 에로 가는 통로이며 다른 하나는 프로쎄스 Y 에서 X 로 가는 통로이다. 정확히 4개 통보문이 매개 통로들에서 이동중에 있고 X 와 Y 가 다 수신을 실행하기전에 그 이상의 송신을 시도한다면 두 프로쎄스는 중단되고 교착이 발생한다.

체계에 있는 매개 쌍의 프로쎄스들사이에서 이동중에 있을 통보문들의 수에 웃한계를 설정할수 있으면 이때 명백한 예방전략은 이 모든 통로들에 필요한만큼 많은 완충기홈들을 배정하는것이다. 이것은 극히 비경제적일수 있다. 요구항목들을 사전에 알수 없거나 웃한계에 기초한 배정이 너무 비경제적인것으로 생각된다면 그때에는 배정을 최대로 활용하기 위한 판정수법술이 요구된다. 이러한 문제는 일반적인 경우에 해결될수 없다는 것을 알수 있다. 이러한 상황을 모방하기 위한 몇가지 경험적전략들이 [BARB 90]에서 제안된다.

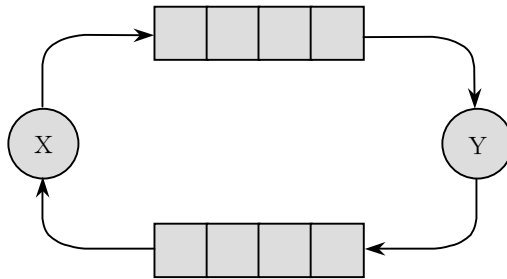



그림 14-19. 분산체계에서 통신교착

요약, 기본용어 및 복습문제

분산조작체계는 프로쎄스이주를 지원할수 있다. 이것은 목적기계에서 실행할 프로쎄스를 위하여 한 기계에서 다른 기계에로 프로쎄스의 많은 중요상태들을 이송하는것을 말한다. 프로쎄스이주는 부하평형과 통신동작을 최소화하는것에 의한 성능개선, 사용률의 증가 또는 프로쎄스들이 전용화된 원격기지들에 접근하도록 하는데 사용될수 있다.

분산체계에서 전역상태정보를 확립하고 자원쟁탈을 해소하며 프로쎄스들을 조종하는 것이 중요하다. 통보문전송에서 예측할수 없는 다양한 시간지연때문에 서로 다른 프로쎄스들이 사건이 발생한 순서를 공동으로 인정한다는것을 보증하는데 주의를 돌려야 한다.

분산체계에서 프로쎄스관리는 호상배제를 집행하고 교착에 관한 처리를 하는 기구들을 포함한다. 두 경우에 단일체계에서보다 문제가 더 복잡하다.

기본용어

통로	퇴거	선취이송
분산교착	전역상태	프로쎄스이주
분산호상배제	비선취이송	순시상기록

복습문제

1. 프로쎄스이주를 실현해야 할 근거를 설명하시오..
2. 프로쎄스이주기간에 프로쎄스주소공간이 어떻게 조종되는가?
3. 선취 및 비선취프로쎄스이주의 동기는 무엇인가?
4. 진짜전역상태를 결정하는것은 왜 불가능한가?
5. 집중형알고리듬이 진행하는 분산호상배제와 분산형알고리듬이 진행하는 분산호상

12. AES가 3중 DES보다 어떻게 되여 개선된것이라고 볼수 있는가?
13. AES후보자들을 평가하는데서 어떤 평가기준을 사용하는가?
14. 전통적인 암호화와 공개열쇠암호화의 차이는 무엇인가?
15. 용어 공개열쇠, 비공개열쇠, 비밀열쇠의 차이는 무엇인가?

참 고 문 헌

[STAL98]은 이 장의 주제에 대하여 더 상세하게 포괄하고 있다. 암호알고리듬들의 포괄범위에 있어서 [SCHN96]은 기본참고서로 된다. 이것은 지난 15년간에 공개된 실제상 모든 암호알고리듬과 규약을 서술하고 있다. 조작체계문제들에 대한 론의는 [BOLL99], [PILE 97], [SINH97], [SING94]에서 찾아 볼수 있다. [HOFF90]과 [DENN90]에서는 침입자와 비루스들과 관련된 많은 주요론문들을 전재하고 있다. [NACH97]은 항비루스기술의 최근동향에 대하여 설명하고 있다. [SHEL97]과 [SUTT97]에서는 Windows NT보안에 대하여 상세하게 설명하고 있다. 관리와 사용에 초점을 두면서 보안과 관련된 NT내부의 몇가지 특징들을 취급하고 있다.

BOLL99 Gollmann, D. *Computer Security.* New York: Wiley, 1999.

DENN90 Denning, P. *Computers Under Attack: Intruders, Worms, and Viruses.* Reading, MA:Addison-Wesley, 1990.

HOFF90 Hoffman. L., editor. Rogue Programs: Viruses, Worms, and Trojan Horses. New York: Van Nostrand Reinhold, 1990.

NACH97 Nachenberg, C. "Computer Virus-Antivirus Coevolution." *Communications of the A CM.* January 1997.

PFLE97 Pfleeger, C. *Security in Computing.* Upper Saddle River, NJ: Prentice Hall PTR, 1997.

SCHN96 Schneier, B. *Applied Cryptography.* New York: Wiley, 1996. SCHN99 Schneier, B. "The Trojan Horse

SHEL97 Sheldon, T. *Window. NT Security Handbook.* New York: Osborne Mcgraw-Hill, 1997.

SING94 Singhat, M., and Shivaratri, N. *Advanced Concepts in Operating Systems.* New York:McGraw-Hill, 1994.

SINH97 Sinha, P. *Distributed Operating Systems.* Piscataway, NJ: IEEE Press, 1997. SMIT82 Smith, A. "Cache

STAL98 Stallings. W. *Cryptography and Network Security: Principles and Practice, 2nd ed.* Upper Saddle River. NJ: Prentice Hall, 1995.

SUTT97 Sutton, S. *Windows NT Security Guide.* Reading. MA: Addison-Wesley. 1997.

련 습 문 제

1. 26개 자모문자중 4개 문자결합으로 통과암호를 선택한다고 가정하자. 적수가 초당 한번씩 통과암호를 시도할수 있다고 가정하자.
   ㄱ) 매개 시도가 끝날 때까지는 적수에 대한 반결합이 전혀 없다고 가정하면 정확한 통과암호를 발견하는 예상시간은 얼마인가?

련 습 문 제

1. 제1절에 있는 프로쎄스이주전략에 관한 소절에서 소각방책을 서술하였다.

   ㄱ) 원천지의 견지에서 보면 어느 전략이 소각과 공통점이 있는가?

   ㄴ) 목적지의 견지에서 보면 어느 전략이 소각과 공통점이 있는가?

2. 그림 14-9에서 q가 비록 P_3에 a보다 먼저 도착할지라도 4개의 모든 프로쎄스는 두 통보문에 $\{a, q\}$순서를 할당할것을 주장한다. 주장의 진실성을 증명하기 위하여 알고리듬을 사용해 보시오.
3. Lamport의 알고리듬에서 P_i가 응답 통보문의 송신 그자체를 보관할수 있는 어떤 사정이 있는가?
4. [RICA81]의 호상배제알고리듬에서

   ㄱ) 호상배제가 수행된다는것을 증명하시오.

   ㄴ) 통보문이 송신된 순서로 도착하지 않으면 알고리듬은 림계구역들이 자기의 요청순서로 실행된다는것을 보증하지 않는다. 고갈이 가능한가?

5. 통표넘기기호상배제알고리듬에서는 분산대기렬알고리듬에서와 같이 시계들을 재설정하고 편차를 수정하기 위하여 시간찍기기능을 사용하는가? 그렇지 않다면 시간찍기의 기능은 무엇인가?
6. 통표넘기기호상배제알고리듬에서 다음과 같은것을 증명하시오.

   ㄱ) 호상배제를 보증한다.

   ㄴ) 교착을 회피한다.

   ㄷ) 공평하다.

7. 그림 14-11 ㄴ에서 두번째 행이 request(j) = t를 왜 쉽게 읽을수 없는가를 설명하시오.

제 7 편. 보안

제 7 편의 중심

만능적인 전자결합성, 비루스와 해커, 전자도청과 전자협잡의 현시대에 보안은 중심론점으로 되였다. 두가지 경향이 출현하여 이 부분에 대한 관심이 사활적으로 높아 졌다. 첫째로,콤퓨터체계와 망을 통한 상호결합의 폭발적인 성장은 이 체계들을 사용하여 보관하고 통신한 정보에 대한 조직들과 개인들의 의존성을 증가시켰다. 이것은 자료와 자원들이 로출되지 않게 보호하고 자료와 통보문들의 인증성을 담보하며 망에 의한 공격으로부터 체계들을 보호해야 할 필요성을 높여 주었다. 둘째로, 암호조작성과 콤퓨터보안에 대한 학문이 성숙되여 보안을 강화할 실용적이고 쉽게 사용할수 있는 응용프로그람들이 개발되고 있다.

제 7 편의 안내

제 15 장. 보안

제 7 편(제 15 장)에서는 조작체계보안과 콤퓨터보안에 대하여 개괄한다. 먼저 보안위협들에 대하여 개괄한다. 다음 콤퓨터보호기구들을 고찰한다. 그다음에 침입자들 즉 권한받지 못한 사용자들 또는 권한받지 못한 동작들을 수행하려고 하는 권한받은 사용자들의 위협에 대항하는 방법들을 론의한다. 다음에 잘 알려 져 있고 가장 해로운 형의 위협들중의 하나인 비루스들을 고찰한다. 이 편에서는 또한 믿음직한 체계라고 하는 콤퓨터보안설계의 포괄적인 방법을 고찰한다. 다음으로 망보안의 기초를 서술한다.

제 15 장. 보 안

콤퓨터보안은 넓은 령역으로서 물리적 및 관리적인 조종들과 자동조종을 포함한다. 이 장에서는 자동화된 보안도구들만을 고찰한다. 그림 15-1에서는 이 도구들의 대응령역을 보여 주고 있다. 콤퓨터통신설비들에 직면한 위협의 종류들을 조사하는것부터 시작한다. 이 장의 많은 부분에서는 보안을 강화하는데 사용할수 있는 특징적인 도구들을 취급한다. 제2절에서는 기억기와 자료를 비롯하여 여러가지 콤퓨터자원의 보호에 기초한 콤퓨터보안의 전통적인 방법들을 취급한다. 다음은 이 보호기구들을 극복하려고 시도하는 사람들에 의한 위협을 고찰한다. 다음절에서는 비루스와 그와 류사한 기구들로부터 제기된 위협을 조사한다. 다음은 믿음직한 체계의 개념을 취급한다. 끝으로 이 장의 부록에서는 많은 보안응용프로그람들에서 사용되고 있는 기본도구인 암호화를 소개하였다.

제 1 절. 보 안 위 협

현재 있는 보안에 대한 위협의 형태들을 리해하기 위하여 보안요구를 정의할 필요가 있다. 콤퓨터 및 망보안은 네가지 요구들을 제기한다. 즉

- **기밀성 :** 콤퓨터체계안의 정보를 오직 승인된 단체들만 읽도록 접근할것을 요구한다. 이 형태의 접근은 대상을 단순히 접근시키는것을 비롯하여 인쇄, 연시 및 그밖의 여러가지 형태의 접근을 포함한다.
- **완정성 :** 콤퓨터체계자원들이 승인된 단체들에 의해서만 변경될것을 요구한다. 변경에는 쓰기, 자료변경, 상태변경, 삭제 그리고 창조가 있다.
- **리용성 :** 콤퓨터체계자산들이 승인된 당사자들에게 리용될수 있어야 한다.
- **인증성 :** 콤퓨터체계가 사용자의 신분을 확인할수 있게 할것을 요구한다.

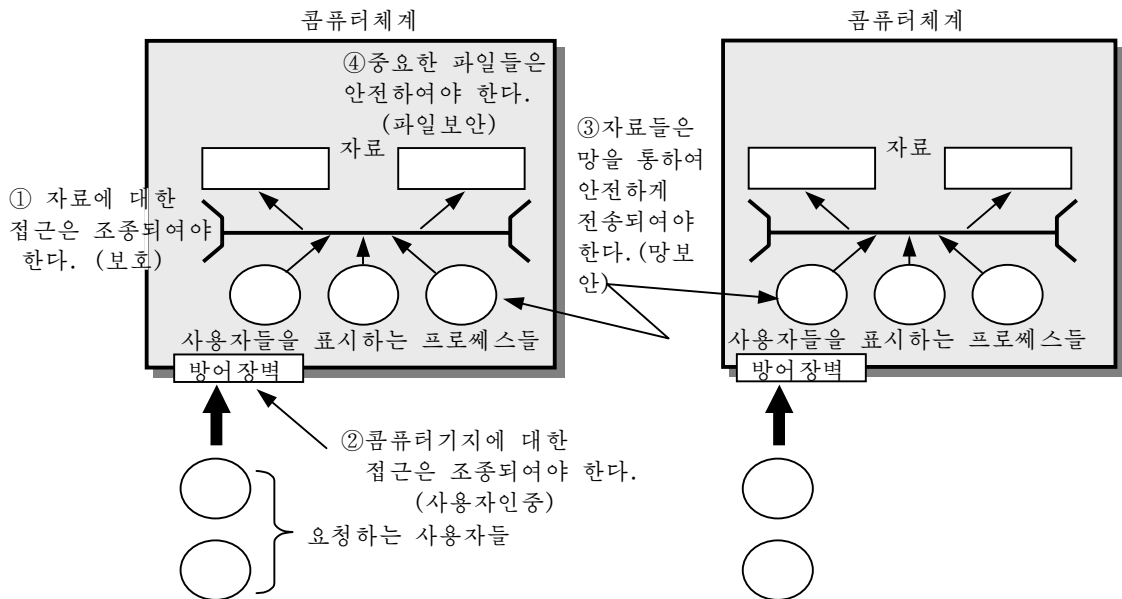



그림 15-1. 체계보안의 령역[MAEK87]

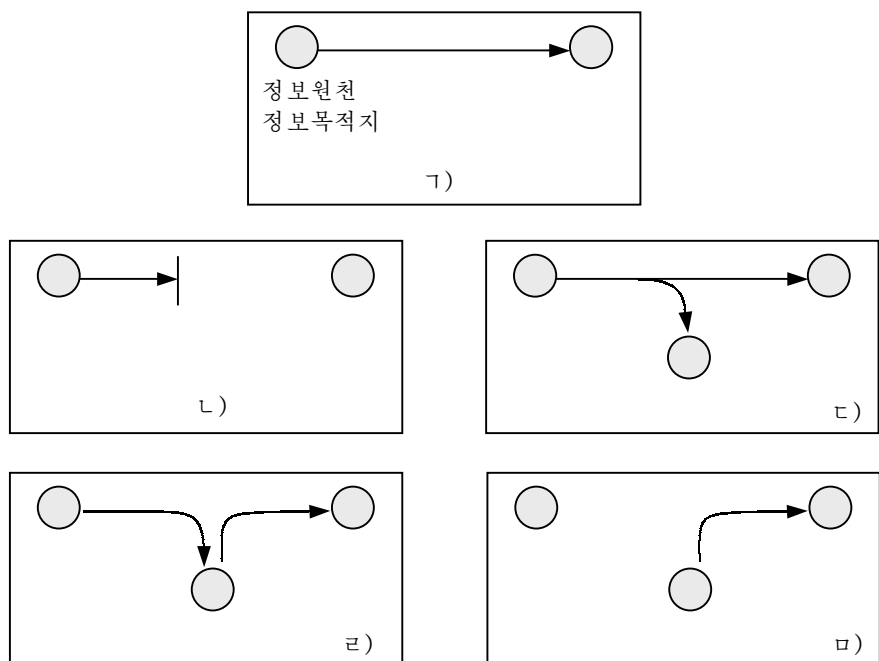



그림 15-2. 보안위협들
ㄱ-보통흐름, ㄴ-중단, ㄷ-가로채기, ㄹ-변경, ㅁ-날조

위협의 형태

콤퓨터체계 또는 망보안에 대한 공격형태들은 콤퓨터체계의 정보제공기능을 고찰함으로써 가장 잘 특징 지을수 있다. 일반적으로 파일 또는 주기억령역과 같은 원천으로부터 또다른 파일 또는 사용자 등의 목적지에로의 정보흐름이 있다. 이 정상적인 흐름이 그림 15-2 ㄱ에 묘사되여 있다. 그림의 나머지 부분은 공격의 네가지 일반종류들을 보여주고 있다.

- **중단 :** 체계의 자산을 파괴하거나 쓸모 없게 또는 사용할수 없게 한다. 이것은 **사용성**측면에서의 공격이다. 실례로서는 하드디스크와 같은 하드웨어부분의 파괴, 통신선의 차단 또는 파일관리체계의 파괴같은것을 들수 있다.
- **가로채기 :** 승인되지 않은 단체가 자산에 접근한다. 이것은 **기밀성**측면에서의 공격이다. 승인되지 않은 단체로는 사람, 프로그람 또는 콤퓨터가 될수 있다. 실례로 망안의 자료를 포착하기 위한 도청과 파일 또는 프로그람들의 비법적인 복사 등을 들수 있다.
- **변경 :** 승인되지 않은 단체가 자산에로의 접근을 얻을수 있을뿐아니라 자산에 간섭한다. 이것은 **완정성**측면에서의 공격이다. 실례로서는 자료파일값들의 변화, 프로그람이 다르게 집행되도록 프로그람의 교체 그리고 망에서 전송되는 통보문들의 내용을 변화시키는것 등을 들수 있다.
- **날조 :** 승인되지 않은 단체가 체계안에 객체를 끼워 넣는다. 이것은 **인증성**측면에서의 공격이다. 실례로서는 망안에 가짜 통보문의 삽입 또는 파일에 기록들의 추가 등을 들수 있다.

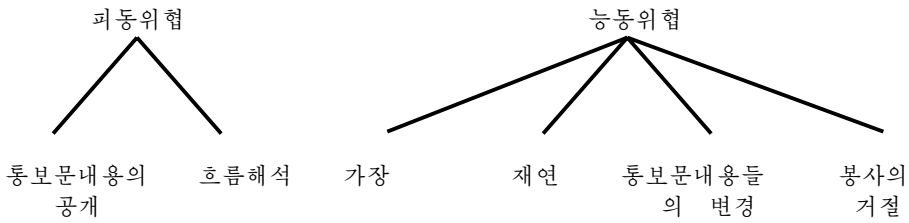



그림 15-3. 능동적인 보안위협과 피동적인 보안위협

콤퓨터체계자산

콤퓨터체계의 자산은 하드웨어, 쏘프트웨어, 자료 그리고 통신선로와 망으로써 구분된다. 그림 15-3과 표 15-1에서는 매개 종류의 자산에 대한 위협의 특징을 보여 주고 있다. 차례로 이것들을 각각 고찰한다.

표 15-1. 보안위협과 자산

	사용성	비밀엄수	완정성/인증성
하드웨어	장치가 도난당하거나 쓸수 없게 되여 봉사를 거부한다.		
쏘프트웨어	프로그람들이 삭제되여 사용자들에 대한 접근을 거부한다.	쏘프트웨어의 비법적인 복사가 진행된다.	작업프로그람이 변경되여 실행시 실패하거나 맹목적인 과제를 수행한다.
자료	파일들이 삭제되여 사용자들에 대한 접근을 거부한다.	자료의 승인되지 않은 읽기가 진행된다.통계자료의 분석은 그 밑에 있는 자료들을 폭로한다.	현존파일들이 변경되거나 새로운 파일들이 날조된다.
통신선로	통보문들이 파괴되거나 삭제된다. 통신선로나 망들이 쓸모 없게 된다.	통보문들이 읽어 진다. 통보문들의 흐름패턴이 관찰된다.	통보문들이 변경,지연,재배렬 또는 중복된다. 가짜 통보문들이 날조된다.

하드웨어

콤퓨터체계의 하드웨어에 대한 주요위협은 사용성의 령역에 있다. 하드웨어는 가장 공격을 받기 쉽고 자동검사를 받기가 가장 힘들다. 위협들에는 절도뿐아니라 우연적 및 고의적인 파괴들이 속한다. 개인용콤퓨터들과 워크스테이션들이 늘어 나고 워크스테이션과 국부망의 사용이 증가함에 따라 이 령역에서 손실위험이 증대되고 있다. 이 위협들에 대처하기 위하여 물리적이며 관리적인 보안수단들이 요구된다.

쏘프트웨어

조작체계, 편의프로그람들 그리고 응용프로그람들은 콤퓨터체계하드웨어가 업무와 개인들에게 쓸모 있도록 하는것들이다. 몇가지 명백한 위협들을 고찰할 필요가 있다.

쏘프트웨어에 대한 주되는 위협은 사용성측면에서의 공격이다. 쏘프트웨어 특히 응용쏘프트웨어는 삭제하기 대단히 쉽다. 쏘프트웨어는 또한 교체할수 있고 그것이 손실되여 쓸모 없게 될수도 있다. 쏘프트웨어의 가장 최근의 판본의 여벌작성을 비롯한 세심한

쏘프트웨어구성관리는 높은 사용성을 유지할수 있게 한다. 대처하기 보다 힘든 문제는 쏘프트웨어가 변경되여 기능은 높지만 본래와 다르게 거동하는 프로그람으로 되여 버리는것이다. 콤퓨터비루스들과 그와 관련한 공격은 이런 범주에 속하든지 이 장의 마감에서 취급한다. 맨 마지막문제점은 쏘프트웨어보안이다. 일부 대응책이 있지만 대체로 쏘프트웨어의 비법적인 복사에 대한 문제들은 원만히 해결되지 못하고 있다.

자료

하드웨어와 쏘프트웨어보안은 대표적으로 콤퓨터쎈터전문가들의 관심사로 되거나 개인용콤퓨터사용자들의 개별적인 관심사로 되고 있다. 훨씬 보다 광범히 고찰되는 문제는 자료보안인데 그것은 개인, 집단, 업무조직들에 의해 조종되는 파일들과 다른 형태의 자료들을 포함한다.

자료와 관계되는 보안의 관심사는 넓으며 사용성, 기밀성, 완정성의 령역이 다 들어간다. 사용성의 경우 관심사는 자료파일의 파괴와 관련되는것인데 그것은 우연적으로 또는 고의적으로 일어 난다.
물론 기밀성과 관계되는 관심사는 자료파일들과 자료기지들의 비법적인 읽기로서 이 령역은 보다 많은 연구사업의 주제로 되고 있으며 콤퓨터보호의 그 어떤 다른 령역보다도 더 많은 노력이 기울여 지고 있다. 명백치 않은 기밀성위협은 종합적인 즉 함축된 정보를 주는 자료의 분석을 비롯하여 이른바 통계자료기지의 사용에서 나타난다. 추측컨대 함축된 정보는 포괄된 개인들의 비밀을 위협하지 않는다. 그러나 통계자료기지의 사용이 늘어 날수록 개인정보가 루설될 가능성이 커진다. 본질상 망라된 개인들의 특징들은 구체적인 분석을 통하여서만 식별될수 있다. 간단한 실례를 들어 만일 한 표가 응답자 A, B, C 그리고 D 의 수입에 대한 총계를 기록하고 또다른 표가 A, B, C, D 그리고 E 의 수입에 대한 총계를 기록한다면 두 총계의 차이가 E 의 수입으로 될것이다. 이 문제는 자료모임들을 련결시키려는 요구가 증가됨에 따라 악화된다. 많은 경우에 그 문제에 적합한 함축준위에서의 일관성을 위하여 몇개의 자료모임들을 대조하여 보는데 이것은 필요한 함축자료들을 구성하는 과정에 요소단위들을 재취급할것을 요구한다. 따라서 개인비밀관계의 대상인 요소단위들은 자료모임들을 처리하는 여러 단계에서 사용가능하다.

끝으로 자료의 완정성은 대부분의 설비들에서 주요관심사로 된다. 자료파일들에 변경이 가해 지면 여러가지 크고작은 손실을 가져 올수 있다.

통신선로와 망

피동적인 공격들은 본질상 전송의 도청이나 전송의 감시이다. 적수의 목적은 전송되는 정보를 획득하는것이다. 여기에는 두가지 형태의 공격 즉 통보문내용의 공개와 흐름해석이 포함된다.

통보문내용의 공개는 쉽게 리해된다. 전화대화, 전자우편통보문 그리고 전송된 파일은 중요하거나 비밀정보를 포함할수도 있다. 이러한 전송의 내용들을 적들이 알지 못하게 하여야 한다.

두번째 피동적인 공격 즉 흐름해석은 보다 포착하기 어렵다. 적들이 통보문을 획득하였다고 하여도 그 통보문으로부터 정보를 알아낼수 없도록 내용들을 가리우는 어떤 방법을 알고 있다고 하자. 정보를 가리우는 일반적인 수법이 곧 암호화이다. 만일 적소에 암호화에 의한 보호대책을 취했다고 하면 적들은 여전히 이 통보문패턴을 관찰할수 있다. 적은 통신하는 가입자들의 위치와 신원을 결정할수 있고 교환되는 통보문들의 빈도와 길이를 고찰할수 있다. 이 정보는 진행되고 있는 통신의 특징을 추출하는데서 쓸모가 있을수 있다.

피동적공격들은 그것들이 그 어떤 자료의 변경도 포함하지 않으므로 검출해내기가 매우 힘들다. 그러나 이 공격들에서의 성과를 막을수 있다. 따라서 피동적인 공격의 취급에서 중점을 두는것은 검출이 아니라 예방이다.

공격의 두번째 주요형태는 능동공격들이다. 이 공격들은 자료흐름의 어떤 변경 또는 틀린 자료의 생성을 포괄하며 네가지 부류 즉 통보문의 가장, 재연, 변경 그리고 봉사거절로 갈라 질수 있다.

가장은 한 실체가 다른 실체인것처럼 보이게 할 때 일어 난다. 가장공격은 보통 능동적인 공격의 다른 형태들중의 하나를 포괄한다. 례를 들어 인증순차들은 유효한 인증순차가 일어난후에 포착되고 재연될수 있고 따라서 몇가지 특권들을 가진 승인받은 실체가 그러한 특권들을 가진 실체를 흉내냄으로써 여분의 특권들을 얻을수 있게 한다.

재연은 자료단위의 피동적인 포착들과 승인받지 못한 효과를 생성시키기 위한 그것의 련이은 재전송을 포함한다.

통보문의 변경은 단순히 진짜 통보문의 어떤 부분이 바뀌여 지거나 통보문들이 승인받지 못한 효과들을 생성시키기 위하여 지연되거나 재배렬되는것을 의미한다. 실례를 들어 통보문 《John Smith 가 비밀파일보고서들을 읽게 하시오.》라는 의미가 《Fred Brown 이 비밀파일보고서들을 읽게 하시오.》라는 의미로 변경된다.

봉사의 거절은 통신설비들의 정상적인 사용이나 관리를 방해하거나 중지시킨다. 이 공격은 특수한 목적을 가질수 있다. 례를 들어 어떤 실체가 특별한 목적지에 향해 진 모든 통보문들을 (례를 들어 보안검열봉사) 금지시킬수도 있다. 봉사거절의 또 하나의 형태는 망을 못쓰게 하거나 성능을 저하시키기 위하여 망에 과부하를 걸어서 전체 망을 와해시키는것이다.

능동적인 공격들은 피동적인 공격들과 반대되는 특징을 가진다. 피동적인 공격들은 검출하기 힘든 반면에 공격성과를 막기 위하여 방어수단들을 사용할수 있다. 다른한편 능동적인 공격들을 합리적으로 막는것은 아주 힘들다. 왜냐하면 그렇게 하자면 모든 통신설비들과 경로들에 대하여 합리적인 물리적보호를 해야 하기때문이다. 그대신에 목적은 그것들을 검출하고 그것들에 의하여 일어 나는 임의의 와해 또는 지연들로부터 회복하는것이다. 왜냐하면 검출이 각이한 효과를 가지며 그것이 또한 공격을 예방하는데 기여할수 있기때문이다.

제 2 절. 보 호

다중프로그람처리의 도입은 사용자들속에 자원들을 공유할수 있는 가능성을 주었다. 이 공유에는 처리기뿐아니라 다음의것도 포함된다. 즉

- 기억기
- 디스크와 인쇄기와 같은 입출력장치들
- 프로그람들
- 자료

이 자원들을 공유할수 있는 능력은 보호에 대한 요구를 끌어 들이였다. [PFLE 97]은 조작체계가 다음의 범위에 따라 보호를 할수 있다는것을 지적하였다. 즉

- **비보호 :** 이것은 중요한 수속들이 서로 다른 시간에 실행되고 있을 때 적합하다.
- **고립 :** 이 방법은 매 프로쎄스들이 공유나 통신이 없이 다른 프로쎄스들로부터 따로따로 동작한다는것을 의미한다. 매 프로쎄스는 자기자신의 주소공간, 파일 그리고 다른 객체들을 가진다.
- **완전공유 또는 비공유 :** 객체(실례로서 파일 또는 기억기토막)의 소유자는 그것들을 공개 또는 비공개형으로 선언한다. 전자의 경우에 임의의 프로쎄스가 객체

에 접근할수 있다. 후자의 경우에는 오직 소유자의 프로쎄스들만이 그 대상에 접근할수 있다.

- **접근제한에 의한 공유 :** 조작체계는 특정한 사용자에 의한 특정한 객체에로의 매개 접근허용을 검열한다. 조작체계는 오직 승인받은 접근만이 일어 나도록 하는 사용자와 객체들사이의 방어장벽 또는 문지기로서 작용한다.
- **동적자격에 의한 공유 :** 객체에 대한 공유권한 등의 동적인 창조를 허락하도록 접근조종의 개념을 확장한다.
- **객체의 사용제한 :** 이 형태의 보호는 한 객체에 대한 접근뿐아니라 그 객체에 대한 사용까지도 제한한다. 실례를 들어 사용자는 중요한 문서를 보는것은 허락되여도 그것을 인쇄하는것은 허락할수 없다. 또하나의 실례로서 사용자는 통계적인 총계를 이끌어 내기 위하여 자료기지에로의 접근은 허용되지만 특정한 자료값들을 결정하는것은 허용되지 않는다.

앞에서 본 항목들은 대략 실행하기 어려운 순서로 제시하였는데 이것은 그것들이 보장하는 보호의 정밀도가 증가하는 순서로도 된다. 주어 진 조작체계는 서로 다른 객체들, 사용자들 또는 응용프로그람들을 위한 서로 다른 등급의 보호를 제공할수 있다.

조작체계는 개별적인 사용자들의 자원을 보호할 필요성과 콤퓨터체계의 사용도를 높이는 공유를 허락할 필요성사이의 균형을 보장할것을 요구한다. 이 절에서는 조작체계가 이 객체들을 위하여 보호를 실시하는 물림새의 일부를 고찰한다.

기억기의 보호

다중프로그람처리환경에서 주기억기의 보호는 필수적이다. 여기서 관심사는 보안이 아니라 실행하고 있는 여러가지 프로쎄스들의 정확한 기능이다. 만일 한 프로쎄스가 우연적으로 다른 프로쎄스의 기억공간에로 쓰기된다면 후자의 프로쎄스는 정확히 실행될수 없다.

여러가지 프로쎄스의 기억공간분리는 가상기억기방식에 의하여 쉽게 달성할수 있다. 토막파일들, 페지파일들 또는 두가지 방법의 결합이 주기억기관리의 효과적인 수단을 준다. 만일 완전한 고립이 요구되면 조작체계는 단순히 매개 토막이나 페지가 그것이 할당되는 프로쎄스에 의해서만 접근될수 있도록 보증하여야 한다. 이것은 페지와 토막표들에 중복되는 입구자료들이 없도록 요구함으로써 쉽게 달성된다.

만일 공유가 허락되면 같은 토막이나 페지는 한개이상의 표에 출현할수도 있다. 이 형태의 공유는 토막화 또는 토막화와 페지화의 결합형을 지원하는 체계에서 가장 쉽게 이루어 진다. 이 경우에 토막구조는 응용프로그람에 접근할수 있으며 응용프로그람은 개별적토막들의 공유 또는 비공유를 선언할수 있다. 순수한 페지환경에서는 두 형태의 기억기의 차이를 구별하기가 보다 힘들다. 왜냐하면 기억기구조가 응용프로그람에 대하여 투명하기때문이다.

기억기보호를 위하여 제공될수 있는 하드웨어지원의 실례로서 OS/390 이 실행하는 IBM SYSTEM/390 계렬의 기계를 들수 있다. 주기억기에서 매 페지프레임과 관련된것은 7bit 기억조종열쇠인데 그것은 조작체계에 의하여 설정될수 있다. 비트들중 2 개는 프레임을 차지하고 있는 페지가 참조되여 변화되였는가를 표시한다. 즉 이것들은 페지치환알고리듬에 사용된다. 나머지 비트 즉 4bit 접근조종열쇠와 하나의 불러내기보호비트는 보호물림새에 사용된다. 기억기에 대한 처리기의 참조들과 DMA 입출력기억기참조들은 그 페지에 접근하는 허락을 얻기 위하여 맞는 열쇠를 사용하여야 한다. 불러내기보호비트는 접근조종열쇠를 쓰기들 또는 읽기와 쓰기들에 적용하겠는가를 표시한다. 처리기에

는 프로그람상태단어(PSW)가 있는데 그것에는 현재 집행중에 있는 프로쎄스와 관련되는 조종정보들이 들어 있다. 이 단어안에 포함된것은 4bit PSW 열쇠이다. 프로쎄스가 페지에 접근하거나 페지상에서 DMA 조작을 시작하려고 할 때 PSW 열쇠는 접근코드와 비교된다. 코드들이 일치할 때에만 쓰기조작이 허용된다. 만일 불러내기비트가 설정되여 있으면 PSW 열쇠는 읽기조작을 위한 접근코드와 일치해야 한다.

사용자지향접근조종

자료처리체계에 접근하기 위하여 주어 진 수단들을 두가지 즉 사용자와 관련된것과 자료들과 관련된것들로 분류한다.

사용자에 의한 접근조종을 때때로 인증이라고 부른다. 이 용어가 현재 통보문인증의 뜻에서 널리 사용되고 있기때문에 여기서는 쓰지 않는다. 그러나 독자들은 이 용어가 사용되는것을 문헌에서 볼수 있을것이다.

공유된 체계나 봉사기의 접근조종을 위한 가장 일반적인 수법은 사용자가입신청인데 이것은 사용자식별자(ID)와 통과암호를 요구한다. 체계는 오직 사용자식별자(ID)가 체계에 알려 져 있고 사용자가 해당 ID 를 가진 체계와 관련된 통과암호를 알 때에만 사용자가 가입하도록 한다. ID/암호체계는 믿음성이 매우 낮은 사용자접근조종방법이다. 사용자는 자기들의 통과암호를 잊어 버릴수 있으며 우연적으로 또는 의식적으로 자기의 통과암호를 루실할수 있다. 해커들은 체계호출과 체계관리자와 같은 특정의 사용자들에 대한 ID 들을 알아 내는데 매우 능숙해 졌다. 결과 ID/통과암호파일은 침입공격의 목표로 된다. 제 3 절에서 그 대책안을 론의한다.

분산환경에서 사용자접근조종은 집중화될수도 있고 분산화될수도 있다. 집중형방법에서 망은 가입봉사를 주어 누구에게 망을 사용하도록 허락하며 그 사용자가 누구와의 련결을 허락받게 하겠는가를 결정한다.

분산형사용자접근조종은 망을 많은 투명한 통신련결로써 취급하며 보통 가입수속은 목적하는 가입자에 의하여 수행된다. 물론 망에서 통과암호를 전송하는데서도 보안은 여전히 관심사로 된다.

많은 망들에서 2 준위의 접근조종을 사용할수 있다. 개별가입자들에게는 가입자특정의 자원들과 응용프로그람들을 보호하기 위하여 가입기능들이 제공될수 있다. 또한 승인받은 사용자들에게 망접근을 제한하기 위하여 망은 보호기능을 제공할수 있다. 2 준위의 기능은 현재 망이 전혀 다른 종류의 가입자들과 접속하고 단순히 말단-가입자접근의 편리한 수단을 제공하는 일반 경우에 필요하다. 보다 전일적인 가입자망에서는 집중된 접근방책이 망조종쎈터에 취해 져야 한다.

자료지향접근조종

가입에 성공한후 사용자는 가입자들과 응용프로그람들중의 한개 또는 모임에 접근하는것이 허락된다. 이것은 일반적으로 자료기지안에 기밀자료들을 포함하고 있는 체계에서는 불가능하다. 사용자접근수속을 통하여 사용자는 체계에 인식된다. 매개 사용자와 관련하여 허용할수 있는 조작과 파일접근을 규정하는 도식이 있을수 있다. 조작체계는 사용자도식에 기초한 규칙들을 실시하여야 한다. 그러나 자료기지관리체계는 특정한 레코드들 지어 레코드들의 부분들에 대한 접근도 조종해야 한다. 실례로 관리부의 누군가에게 회사종업원목록에는 접근하도록 허용될수 있지만 선택된 개별적인 사무원정보에는 접근할수 없다. 그 문제점은 세부로 들어 갈수록 더 많다. 조작체계가 보안검사가 더는 없는 조건에서 파일접근과 응용프로그람사용을 사용자에게 허락한다면 자료기지관리체계는 매개 개별적인 접근시도에 대한 결정을 하여야 한다. 그 결정은 사용자의 신원확인뿐

아니라 접근되는 자료의 특정한 부분에도 관계되며 지어 사용자에게 이미 루설된 정보에도 관계된다.

파일 또는 자료기지관리체계가 실행하는 접근조종의 일반모형은 **접근행렬**의 모형이다(그림 15-4 ㄱ, [SAND94]의 그림에 기초). 모형의 기본요소들은 다음과 같다. 즉

- **주동체 :** 객체들에 접근할수 있는 실체. 일반적으로 주동체의 개념은 프로쎄스의 개념과 동등하다. 임의의 사용자나 응용프로그람은 실제로 그 사용자 또는 응용프로그람이 제시하는 프로쎄스에 의하여 객체에 접근한다.

	파일 1	파일 2	파일 3	파일 4	회계 1	회계 2
사용자 A	Own R W		Own R W		조사 대변	
사용자 B	R	Own R W	W	R	조사 차변	조사 대변
사용자 C	R W	R		Own R W		조사 차변

ㄱ)

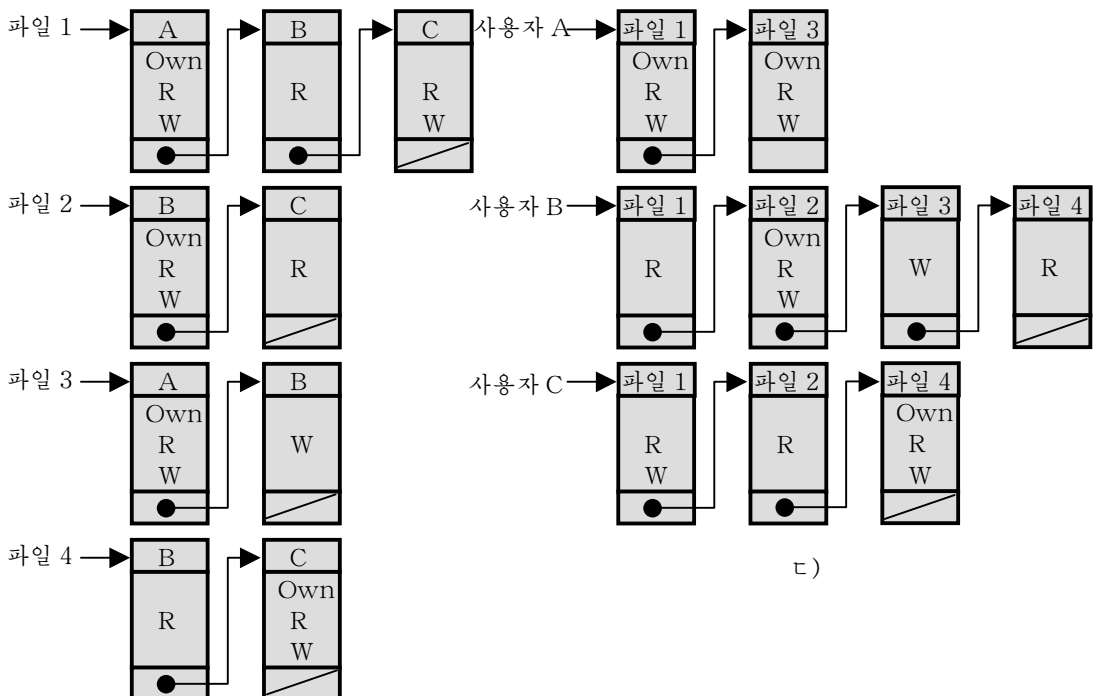



그림 15-4. 접근조종구조들의 실례
ㄱ-접근행렬, ㄴ-부분파일들에 대한 접근조종목록,
ㄷ-ㄱ부분파일들에 대한 자격목록

- **객체 :** 접근이 조종되는것. 실례를 들어 파일들, 파일의 부분들, 프로쎄스들 그리고 기억기토막들이 속한다.
- **접근권 :** 객체가 주동체에 따라 접근되는 방법. 실례를 들어 읽기, 쓰기, 집행과 같은것이 있다.

행렬의 한 차원은 자료에 접근할수 있는 식별된 주동체들로 이루어 져 있다. 전형적으로 이 목록은 말단들, 가입자들, 사용자들을 대신하는 응용들에 관하여 접근이 통제상태에 있지만 개별적사용자 또는 사용자집단으로 구성된다. 다른 차원은 접근될수 있는 객체들을 목록화한다. 가장 큰 준위의 세부에서 객체들은 개별적인 자료마당들일수 있다. 레코드, 파일 지어는 옹근자료기지와 같은 보다 총체적인 그룹이 행렬의 객체일수 있다. 행렬안의 매 입구자료는 그 객체에 대한 주동체의 접근권을 표시한다.

실천에서 접근행렬은 드물며 두 방법들중의 한가지로 분해하는 방법으로 실현된다. 행렬은 렬들로 분해되여 **접근조종목록**들을 산생할수 있다(그림 15-4 ㄴ). 따라서 매개 객체에 대하여 접근조종목록은 사용자들과 그들에게 승인된 접근권들을 목록으로 작성한다. 접근조종목록은 기정 또는 공개입구자료을 포함할수 있다. 이것은 특수한 권한을 가진다고 명확히 목록화되여 있지 않는 사용자들이 기정의 권한들을 가지도록 한다. 목록의 구성요소는 사용자그룹들뿐아니라 개별적사용자들도 포함할수 있다.

행에 의한 분해는 **자격표**들을 생성한다(그림 15-4 ㄷ). 자격표는 사용자에게 승인된 객체들과 조작들을 정의한다. 매 사용자는 여러개의 표들을 가지며 그것들을 다른사용자들에게 빌려 주거나 넘겨 주는 특권을 가질수도 있다. 표들이 체계에 분산될수 있기때문에 그것들은 접근조종목록보다 더 큰 보안문제를 제기한다. 특히 표는 날조될수 없어야 한다. 이렇게 하기 위한 한가지 방법은 조작체계가 사용자들을 대표하는 모든 표들을 가지도록 하는것이다. 이 표들은 사용자들이 접근할수 없는 기억기령역에 유지되여야 한다.

자료지향접근조종을 위한 망의 고려대상은 사용자지향접근조종의것들과 류사하다. 만일 특정한 사용자들만 자료의 일정한 항목들에 접근할수 있도록 허용한다면 승인받은 사용자들에게 전송되는동안 그 항목들을 보호하기 위하여 암호화가 필요할수 있다. 전형적인 경우 자료접근조종은 분산된다. 즉 가입자에 기초한 자료기지관리체계들로 조종된다. 만일 망자료봉사기가 망에 존재하면 자료접근조종은 망기능으로 된다.

제 3 절. 침 입 자

보안에서 가장 널리 알려 진 두가지 위협들중의 하나는 침입자(다른것은 비루스이다.)로서 일반적으로 해커라고 한다. Anderson[ANDE8a]는 세가지 종류의 침입자들을 밝혔다. 즉

- **가장자 :** 콤퓨터를 사용하도록 승인되지 않은 개인과 합법적인 사용자자격을 도용하여 체계의 접근조종에 침입하는 개인
- **직권람용자 :** 접근이 승인되지 않은 자료나 프로그람 또는 자원들에 접근하는 합법적인 사용자 또는 그러한 접근이 승인되였지만 자기의 특권들을 람용하는 합법적인 사용자
- **비밀사용자 :** 체계감독조종을 빼앗아 조종을 사용하여 회계감시와 접근조종을 회피하거나 회계종합을 억제하는 개인

가장자는 외부사람, 직권람용자는 일반적으로 내부사람 그리고 비밀사용자는 어느쪽이나 될수 있다.

침입자공격에는 가벼운것도 있고 엄중한것도 있다. 가벼운 침입자들속에는 단순히 인터네트를 탐색할것을 원하며 남의것을 보기를 원하는 사람들이 있다. 엄중한 침입자들이란 비밀자료를 읽거나 자료에 대한 승인받지 않은 수정을 하거나 체계를 파괴하려는 사람들이다.

침입수법

침입자의 목표는 체계에 대한 접근권을 얻거나 체계상에서 접근할수 있는 특권들의 령역을 넓히는것이다. 일반적으로 침입자는 보호되여야 할 정보를 얻을것을 요구한다. 많은 경우에 이 정보는 사용자통과암호를 가진 형태로 되여 있다. 사용자의 통과암호를 알면 침입자는 체계에 가입하여 합법적으로 사용자에게 주어 진 모든 특권들을 행사한다.

일반적으로 체계는 매개 승인된 사용자와 통과암호를 관련시키는 파일을 가지고 있어야 한다. 만일 그러한 파일이 보호가 없이 보관되여 있으면 그것에 접근하고 통과암호를 알아 내는것은 쉬운 문제이다. 통과암호파일은 두가지 방법들중 어느 하나로 보호되여야 한다. 즉

- **한방향암호화 :** 체계는 사용자통과암호의 암호화된 형태만을 보관한다. 사용자가 통과암호를 대면 체계는 그 암호를 암호화하여 보관된 값과 비교한다. 실천에서 체계는 보통 통과암호를 암호화기능을 위한 열쇠를 산출하기 위하여 사용하며 고정길이출력이 만들어 지는 한방향변환(가역할수 없는)을 수행한다.
- **접근조종 :** 통과암호파일에 대한 접근은 한번 또는 매우 적은 회수의 회계들로 제한한다.

만일 이 대책들중에서 한가지 또는 두가지가 다 취해 지면 침입자는 통과암호를 알기 위하여 어떤 수고를 하여야 한다. 문헌에 대한 연구와 많은 침입자들과 진행한 상담들에 기초하여 [ALYA90]에서는 통과암호를 알아 내기 위한 수법들을 다음과 같이 보고하였다. 즉

1. 체계에 장비된 표준장부들에서 사용된 기정통과암호를 알려고 시도해 본다. 많은 관리자들은 이 기정값들을 변화시키려고 하지 않는다.
2. 모든 짧은 통과암호들(1～3문자로 된것)을 남김없이 시험해 본다.
3. 체계의 직결사전안의 단어들 혹은 가능한 통과암호들의 목록에 있는 단어들을 시도해 본다. 목록의 실례는 해커광고판에서 쉽게 얻을수 있다.
4. 완전한 이름, 후견인과 어린이이름, 사무실의 그림들과 취미와 관련된 사무실의 책들과 같은 사용자들에 대한 정보를 수집한다.
5. 사용자의 전화번호, 사회보안번호, 방번호를 시도해 본다.
6. 이 상태에 대한 모든 합법적인 면허간판번호들을 시도해 본다.
7. 접근의 우회제한들에 대한 트로이목마(제 4 절에 서술된)를 사용한다.
8. 원격사용자와 가입자체계사이의 선로를 엿듣는다.

첫 6 가지 방법들은 통과암호를 추측하는 몇가지 방법들이다. 만일 침입자가 가입을 시도하여 추측의 정확성을 확인하여야 한다면 그것은 지루한 방법이며 공격수단이 역습을 쉽게 받는다. 실례로서 체계는 세번 통과암호를 준후에 어떤 가입도 거절할수 있고 따라서 침입자는 가입을 다시 시도하기 위하여 가입자가 다시 련결할것을 요구한다. 이러한 정황에서 자그마한 통과암호보다 더 많은것을 시도하는것은 실천적이지 못하다. 침입자는 그러한 조잡한 방법들을 시도하지 않을것이다. 실례로 만일 침입자가 낮은 수준의 특권으로 암호화된 통과암호파일에 접근할수 있다면 그때 전략은 그러한 파일을 포착

한다음 보다 높은 특권들을 주는 적합한 통과암호가 발견될 때까지 여유있게 그 특정한 체계의 암호기구를 사용하는것이다.

추측프로쩨스에 의한 검출을 하지 않고도 많은 추측들을 자동적으로 할수 있고 또 매개 추측들을 확인할수 있는 경우에는 추측공격들이 가능하고 또 대단히 효과적이다. 이 절의 마지막 부분에서 추측공격의 방해법에 대해 언급한다.

앞에서 제시된 일곱번째 방법인 트로이목마는 반격하기가 대단히 힘들수 있다. 접근조종을 우회하는 프로그람의 실례를 [ALUA90]에 소개하고 있다. 낮은 특권사용자는 유희프로그람을 제작하고 그것을 체계운영자들이 자기의 여가시간에 사용하도록 하였다. 프로그람은 실지로 유희를 놀았지만 그 밑바탕에는 암호화되지는 않았어도 접근이 보호되여있는 통과암호파일을 사용자파일에로 복사하기 위한 코드를 포함하였다. 유희프로그람이 조작자의 고특권방식에서 실행되기때문에 통과암호파일에로의 접근권을 얻을수 있다.

제시된 8 번째 공격 즉 선로엿듣기는 물리적보안문제이다. 그것은 련결암호화수법등을 사용하여 반격할수 있다.

여기서는 두가지 기본적인 대책안들 즉 예방과 검출에 대해서만 설명하기로 한다. 예방은 보안목표에 대한 도전이며 항상 힘든 싸움이다. 애로는 방위자가 모든 가능한 공격들을 막기 위하여 노력하여야 하는 반면에 공격자는 방어사슬안의 제일 약한 고리를 찾아 그것을 공격하는데서 자유롭다는것이다. 검출은 공격전이나 후에 그 공격을 알아내는것이다.

통과암호보호

침입자들을 막아 내는 방위의 일선은 통과암호체계이다. 실제로 모든 다중사용자체계들은 이름이나 식별자(ID)뿐아니라 통과암호도 입력할것을 요구한다. 통과암호는 그 체계에 대한 개별적인 가입의 ID를 인증하는데 봉사한다. 그리고 ID는 다음의 방법으로 보안을 보장한다. 즉

- ID는 사용자가 체계에 접근하는것이 합법적인가를 결정한다. 일부 체계들에서는 이미 체계에 기록된 ID를 가진 사람들만 접근하는것이 허락된다.
- ID는 사용자에게 부여될 권한들을 결정한다. 몇명의 사용자들은 조작체계에 의하여 특별히 보호되는 파일을 읽고 기능들을 수행하는 관리자 또는 《고준위사용자》상태에 있을수도 있다. 일부 체계들은 손님 또는 가명의 가입장부들을 가지며 이 가입장부의 사용자들은 다른 사람들보다 더 제한된 다른 권한을 가진다.
- ID는 자유접근조종이라고 부르는 곳에서 사용된다. 실례를 들어 다른 사용자들의 ID가 제시되면 한 사용자는 자기가 소유한 파일들을 다른 사용자들이 읽을수 있도록 허락할수 있다.

통과암호들의 취약성

공격의 특징을 리해하기 위하여 통과암호들이 평문으로 보관되지 않고 UNIX체계상에서 널리 사용되는 방안을 고찰하자. 다음의 절차를 사용하기로 하자(그림 15-5ㄱ). 매개 사용자는 길이가 8문자까지의 통과암호를 선택한다. 이것은 암호화루틴의 입력열쇠로서 봉사하는 56bit값(7bit의 ASCII를 사용하여)으로 변환된다. crypt(3)으로 알려 진 암호화는 DES에 기초하고 있다. DES알고리듬은 12bit《양념[1]》값을 사용하여 변경된다. 전형적으로 이 값은 통과암호가 사용자에게 할당된 시간에 따라 결정된다. 변경된 DES알고리듬은 0의 64bit 블로크로 이루어 진 자료입력과 함께 실행된

---

[1] 암호화할 때 섞는 구절

다. 이 처리는 총 25개 암호화과정에 반복된다. 결과인 64bit출력은 다음에 11문자렬로 분할된다. 양념의 평문사본과 함께 이 암호문통과암호는 대응하는 사용자 ID에 대한 통과암호파일에 보관된다.

양념은 세가지 목적에 봉사한다.

- 그것은 통과암호파일안에서 있을수 있는 암호들의 중복을 막는다. 지어 두 사용자가 같은 통과암호를 선택하였다고 할지라도 그 암호들은 서로 다른 시간에 할당될것이다. 이로부터 두 사용자들의 《확장된》통과암호들은 다르게 될것이다.
- 그것은 사용자에게 두개의 추가적인 문자들을 기억할것을 요구함이 없이 통과암호의 길이를 효률적으로 증가시킨다. 이로부터 가능한 암호들의 수는 4096개의 요소로 증가되여 암호를 추측하기 힘들게 한다.
- 그것은 열쇠전수추측공격의 곤난성을 덜어 줄수 있는 DES 의 하드웨어제품의 사용을 방지한다.

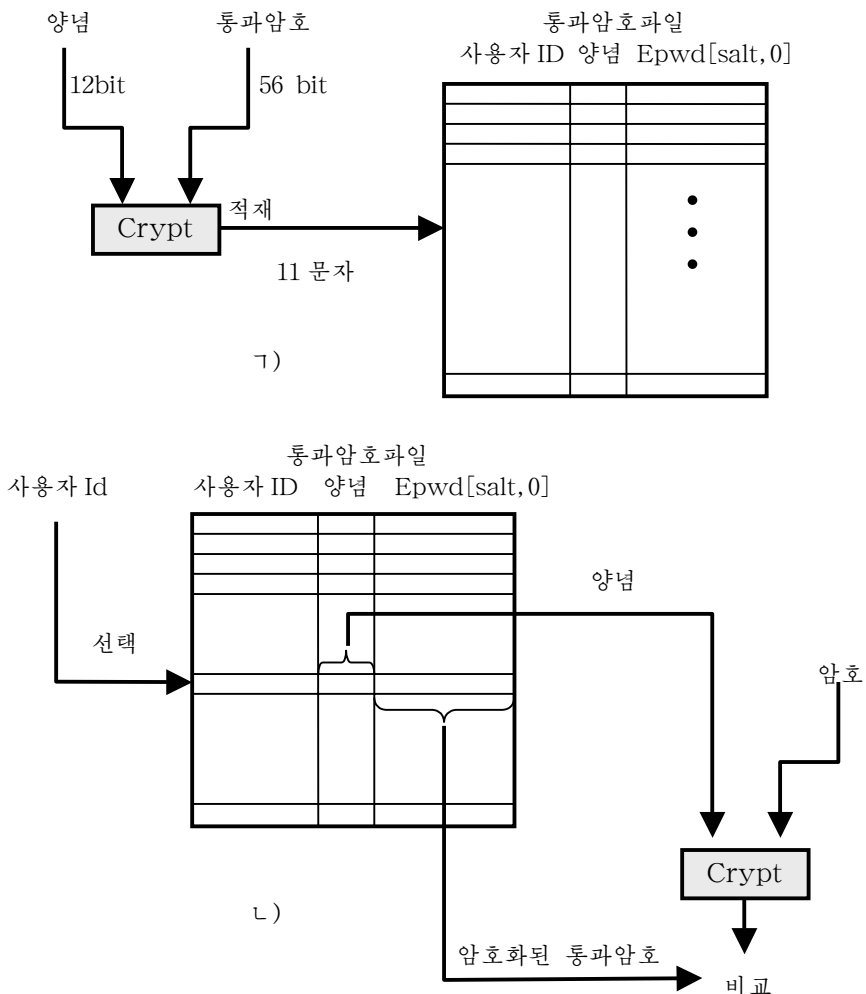



그림 15-5. UNIX 통과암호

ㄱ-새로운 통과암호의 적재, ㄴ-통과암호의 확인

사용자는 UNIX체계에 가입하려고 할 때 ID와 통과암호를 준다. 조작체계는 암호루틴의 입력으로 사용되는 ID에 의하여 통과암호파일에 색인을 붙이며 평문으로 된 양념과 암호화된 통과암호를 검색한다. 만일 결과가 보관된 값과 일치하면 통과암호는 접수된다.

암호루틴들은 추측공격을 단념하도록 설계된다. DES의 쏘프트웨어적실현은 하드웨어적실현들에 비하여 속도가 느리다. 그리고 25번의 반복이 진행되므로 암호화에 필요한 시간을 25배로 증가시킨다. 그러나 이 알고리듬에서는 원래 설계로부터 두가지 변화가 발생하였다. 첫째로, 알고리듬 그자체가 새롭게 실현되면서 고속화된것이다. 실례를 들어 인터네트벌레는 공격 받는 UNIX체계상에 보관된 표준알고리듬보다 더 효률적인 암호화알고리듬을 사용함으로써 매우 짧은 시간동안에 수백개의 통과암호들의 직결암호추측을 할수 있었다. 둘째로, 하드웨어성능은 임의의 쏘프트웨어알고리듬을 빨리 집행할수 있도록 계속 증가하고 있다.

그래서 UNIX통과암호방식에는 두가지 위협들이 존재한다. 첫째로, 사용자는 손님장부를 사용하거나 또는 일부 다른 수단들로 기계에 대한 접근권을 얻고 그 기계에서 통과암호해독자라고 하는 통과암호추측프로그람을 실행한다. 공격자는 적은 량의 자원소비로 가능한 수백수천개의 통과암호들을 검사할수 있다. 게다가 만일 대방의 통과암호파일의 사본을 얻을수 있으면 해독자프로그람은 천천히 다른 기계에서 실행될수 있다. 이것은 공격자가 적당한 시간안에 가능한 수천개의 통과암호들을 다 실행해 볼수 있게 한다.

실례로서 한 통과암호해독자가 1993년 8월에 인터네트에 보고되였다[MADS93]. Thinking Machines Corporation 의 병렬콤퓨터를 사용하여 벡토르장치당 초당 1560개의 암호화의 성능을 실현하였다. 처리마디당 4개의 벡토르장치들로 구성된 경우(표준구성) 이것은 128개 마디의 기계(적당한 규모)상에서 초당 800000개의 암호들을 풀며 1024개 마디의 기계상에서 초당 640만개의 암호들을 푼다.

이 굉장한 추측속도도 아직 공격자가 암호를 찾기 위하여 문자들의 가능한 모든 조합들을 시도하는 우둔한 열쇠전수공격수법을 사용할수 없게 한다. 대신에 통과암호해독자들은 일부 사람들이 쉽게 추측할수 있는 통과암호들을 선택한다는 사실에 의거한다.

일부 사용자들은 통과암호를 선택할 때 불합리하게 짧은 통과암호를 선택한다. Purdue종합대학에서 진행한 연구결과를 표 15-2에 제시하였다. 연구는 대략 7000개의 사용자가입장부를 대상으로 54개의 기계에서 입구되는 통과암호의 길이변화들을 관찰하였다. 거의 3%의 통과암호의 길이가 세문자이거나 그보다 더 짧았다. 공격자는 길이가 3 또는 그보다 더 짧은 모든 가능한 통과암호들을 남김없이 시험하여 공격을 시작할수 있었다. 간단한 대책은 체계에 대하여 6문자보다 더 작은 임의의 통과암호선택을 거부하며 또는 모든 통과암호들이 정확히 길이가 8문자일것을 요구하도록 하는것이다. 많은 사용자들은 그러한 제한에 대하여 동의하지 않는다.

통과암호의 길이는 유일한 문제점이다. 자기들자신이 통과암호를 선택하는것이 허락될 때 많은 사람들이 자기의 이름, 거리이름, 일반사전단어와 같은 추측할수 있는 통과암호를 선택한다. 이것은 통과암호해독자의 일감을 간단하게 만든다. 해독자는 단순히 있을수 있는 통과암호들의 목록으로 통과암호파일을 검사하면 된다. 많은 사람들이 추측할수 있는 통과암호들을 사용하기때문에 그러한 전략은 실제로 모든 체계들에서 성공할수 있다.

추측의 효과성에 대한 한가지 실례가 [KLE190]에서 보고되였다. 여러가지 원천으로부터 저자는 거의 14000개의 암호화된 통과암호들을 포함하는 UNIX통과암호파일들을

수집하였다. 저자가 놀라운것이라고 특징 지은 결과를 표 15-3에서 보여 주고 있다. 모든 통과암호들의 거의 4분의1이 추측되였다. 다음과 같은 단계로 추측이 진행되였다. 즉

1. 사용자이름, 첫머리글자들, 장부이름 그리고 다른 관련되는 개인정보들을 시도한다. 매 사용자에 대하여 모두 130번의 변경된 시도들이 진행되였다.
2. 여러가지 사전들의 자료들을 시도해 본다. 저자는 체계 그자체에 직결되여 있는 사전과 표에서 보여 준바와 같은 여러가지 다른목록들을 비롯하여 60000 단어이상의 사전을 번역하였다.
3. 2 단계로부터 단어들에 여러가지 변경을 가하여 시도한다. 이것은 첫 문자를 대문자 또는 조종문자로 만들기, 전체 단어를 대문자로 만들기, 단어를 뒤집기, 문자《O》를 수자 《0》으로 변화시키기 등을 포함한다. 이 변경들에 의해 또다른 백만개 단어들을 목록에 추가하였다.
4. 3단계에서 고찰되지 않은 2단계의 단어들에 여러가지 대문자변환들을 시도한다. 이렇게 하여 거의 2백만개의 추가적인 단어들을 목록에 첨부하였다.

표 15-2. 고찰된 통과암호길이

길이	개수	총적인 비률 %
1	55	.004
2	87	.006
3	212	.02
4	449	.03
5	1260	.09
6	3035	.22
7	2917	.21
8	5772	.42
합계	13.787	1.0

표 15-3. 13797개 장부들의 실례모임으로부터 해독된 통과암호들[KLE190]

통과암호의 형태	시도한 회수	해독한 건수	해독한 통과암호의 비률 %
사용자/가입자이름	130	368	2.7%
문자렬	866	22	0.2%
수자	427	9	0.1%
중어	392	56	0.4%
장소이름	628	82	0.6%
공통이름	2239	548	4.0%
녀자이름	4280	161	1.2%
남자이름	2866	140	1.0%
특수한 이름	4955	130	0.9%
신화들과 전설들	1246	66	0.5%

표계속

쉑스피어작품이름	473	11	0.1%
체육용어	238	32	0.2%
과학이야기	691	59	0.4%
영화와 배우들	99	12	0.1%
만화	92	9	0.1%
명인	290	55	0.4%
명문장	933	253	1.8%
별명	33	9	0.1%
생물학	58	1	0.0%
체계사전	19683	1027	7.4%
기계이름	9018	132	1.0%
련상기호	14	2	0.0%
제임스왕성서	7525	83	0.6%
잡다한 단어들	3212	54	0.4%
유태단어	56	0	0.0%
소행성	2407	19	0.1%
총계	62727	3340	24.2%

이러한 시험은 대략 3백만개의 단어들을 포함하였다. 앞에서 제시된 최고속 Thinking Machines Implementation 을 사용하면 가능한 양념값들을 사용하면서 이 모든 단어들을 암호화하는 시간은 한시간정도이다. 그러한 체계성 있는 연구는 대략 25%의 성공률을 산출하지만 지어 한번의 명중도 체계의 넓은 령역의 비밀을 얻어 내는데 충분하다는것을 명심해야 한다.

접근조종

통과암호공격을 막기 위한 한가지 방법은 통과암호파일에 대한 대방의 접근을 거부하는것이다. 만일 파일의 암호화된 통과암호부분이 특권을 가진 사용자에 의해서만 접근될수 있다면 상대방은 특권을 가진 사용자의 통과암호를 이미 알지 않고는 그것을 읽을수 없다. [SPAF92]는 이 전략에서의 몇가지 약점들을 언급한다. 즉

- 대부분의 UNIX 체계들을 비롯한 많은 체계들은 침입에 민감하다. 일단 공격자는 어떤 수단들에 의해 접근권을 얻었으면 검출의 위험을 감소시키기 위하여 서로 다른 가입접속에 서로 다른 가입장부들을 사용할수 있도록 통과암호들의 집합을 얻으려고 할수 있다. 또는 가입장부를 가진 사용자는 비밀자료들에 접근하거나 체계를 마비시키기 위하여 다른 사용자의 가입장부를 원할수 있다.
- 보호의 돌발사고는 통과암호파일을 읽을수 있도록 내버려 둘수 있으며 이렇게 하여 모든 가입장부들은 루설될수도 있다.
- 일부 사용자들은 다른 보호령역안의 다른 체계들에 가입장부를 가지며 같은 통과암호를 사용한다. 따라서 만일 통과암호가 한 기계상의 누군가에 의하여 읽어 질수 있으면 다른 위치의 기계들이 침입당할수도 있다.

따라서 보다 효과적인 전략은 추측하기 힘든 통과암호들을 사용자들이 선택하도록 하는것이다.

통과암호선택전략

방금 서술된 두 실험들(표 15-2와 15-3)의 교훈은 장치적측면은 론하지 않더라도 많은 사용자들이 추측하기가 지내 간단하거나 지내 쉬운 통과암호를 선택하는것이다. 다른 한편 만일 사용자들에게 우연으로 선택한 8개의 인쇄문자들로 이루어 진 통과암호들이 할당되면 통과암호해독은 사실상 불가능하다. 그러나 대부분의 사용자들이 자기들의 통과암호를 기억하는것도 거의 불가능할수 있다. 적당히 기억할수 있는 문자들의 렬로 통과암호령역을 제한한다고 해도 령역의 크기가 너무 커서 실제적인 해독을 수행할수 없다. 목적은 사용자가 기억할수 있는 통과암호를 선택하도록 하면서도 추측할수 있는 통과암호들을 없애도록 하는것이다. 네가지 기본수법이 사용되고 있다.

- 사용자교육
- 콤퓨터로 만든 통과암호
- 통과암호거부검사
- 통과암호합격검사

사용자들에게 추측하기 힘든 통과암호들의 중요성이 강조될수 있으며 강력한 통과암호들을 선택하기 위한 지침들을 보장할수 있다. 이 **사용자교육**전략은 대부분의 설치장소들 특히 큰 사용자집단 또는 변동이 많고 심한곳에서는 성공할것 같지 못하다. 많은 사용자들이 그 지침들을 간단히 무시해 버릴것이다. 다른 사람들은 무엇이 강력한 통과암호라고 옳바른 판단을 내리지 못할수 있다. 실례를 들어 많은 사용자들이 단어를 뒤집거나 마지막 문자를 대문자로 하는것이 통과암호를 추측할수 없게 만든다고 (잘못) 믿고 있다.

콤퓨터로 만든 통과암호들 또한 문제점을 가지고 있다. 만일 통과암호들이 사실상 완전히 우연선택한것이라면 사용자들은 그것들을 기억할수 없을것이다. 지어 통과암호가 발음으로 나타낼수 있는것이라고 할지라도 사용자는 그것을 기억하기 힘들수 있으며 어딘가에 기록하고 싶어 할것이다. 일반적으로 콤퓨터로 만든 통과암호방안들은 사용자들이 잘 접수하지 않은 력사를 가지고 있다. FIPS PUB 181 은 최고로 잘 설계된 자동암호발생기들중의 하나를 정의한다. 그 표준안은 방법에 대한 서술뿐아니라 알고리듬의 C 원천코드의 완전한 목록을 포함하고 있다. 그 알고리듬은 단어를 만들기 위하여 발음할수 있는 음절들을 선정하고 그것들을 련결하는것으로 단어들을 발생한다. 란수발생기는 음절들과 단어들을 만들기 위하여 문자들의 무질서한 흐름을 만들어 낸다.

통과암호거부검사전략은 추측할수 있는 통과암호들을 찾기 위하여 자기가 가지고 있는 통과암호해독프로그람을 주기적으로 실행시키는 전략이다. 체계는 추측되는 임의의 통과암호들을 삭제하고 사용자들에게 알려 준다. 이러한 방책은 많은 결함들을 가진다. 우선 만일 일감이 정확히 수행된다면 그것은 자원긴장이다. 통과암호파일을 훔칠수 있는 한정된 상대방이 몇시간 지어 며칠동안 그 과제에 충분한 CPU 시간을 바치기때문에 효과적인 통과암호거부검사자는 매우 불리한 위치에 있다. 게다가 모든 현존통과암호들은 통과암호거부검사자가 자기들을 발견할 때까지 취약한 상태에 남아 있게 된다.

개선된 통과암호보안에 대한 가장 믿음직한 방법은 **통과암호합격검사기**이다. 이 방식에서는 사용자가 자기자신의 통과암호를 선택할수 있다. 그러나 선택의 시점에서 체계는 통과암호가 쓸만한가 아닌가를 알기 위해 검사하며 만일 아니라면 그것을 거부한다.

그러한 검사기들은 체계로부터 충분한 지도를 받으며 사용자들이 사전을 사용한 공격에서 추측될것 같지 않은 상당히 거대한 통과암호공간으로부터 기억할수 있는 통과암호들을 선택할수 있다는 원리에 기초하고 있다.

통과암호합격검사기를 사용하는데서 요령은 사용자접수능력과 강도사이의 균형을 맞추는것이다. 만일 체계가 너무 많은 통과암호들을 거부하면 사용자들은 통과암호들을 선택하는것이 지내 힘들다는 의견들을 제기할것이다. 만일 체계가 무엇을 접수할수 있는가를 정의하기 위하여 어떤 단순한 알고리듬을 사용한다면 이것은 통과암호해독자들에게 자기들의 추측수법을 개선할수 있게 하는 지침을 주는것으로 된다. 이 소절의 나머지 부분에서 통과암호합격검사에 대한 가능한 방법들을 설명한다. 첫번째 방법은 규칙실시를 위한 단순한 체계이다. 실례를 들어 다음의 규칙들을 실시할수 있다.

- 모든 통과암호들은 적어도 8문자길이를 가져야 한다.
- 첫 8 문자들에서 통과암호들은 적어도 대문자, 소문자, 수자들 그리고 구두점들을 하나씩 포함하여야 한다.

이 규칙들은 사용자에 대한 충고와 결합되여야 한다. 비록 이 방법이 사용자들을 단순히 교육하는것보다 더 우월하지만 그것은 통과암호해독자들을 방해하는데 충분하지 못할수 있다. 이 방안은 해독자들이 통과암호를 풀려고 시도하지 못하도록 경계하지만 여전히 통과암호해독을 가능하게 할수 있다.

또하나의 가능한 방법은 단순히 가능한 《나쁜》통과암호들의 거대한 사전을 편집하는것이다. 사용자는 통과암호를 선택할 때 체계는 그것이 검토되지 않은 목록에 있지 않는가를 확인하기 위하여 검사를 한다. 이 방법에는 두가지 문제점들이 있다. 즉

- **공간 :** 사전은 효과적이도록 매우 커야 하는것이다. 실례를 들어 Purdue의 연구[SPAF92]에서 사용된 사전은 30MB공간이상을 차지한다.
- **시간 :** 거대한 사전을 탐색하는데 요구되는 시간은 클수 있다. 게다가 사전단어들의 가능한 바꿔놓기를 검사하기 위하여 그러한 단어들을 많이 사전에 포함시켜 사전을 실지로 크게 만들거나 또는 매개 탐색이 상당한 처리를 진행하여야 한다.

침입검출

가장 좋은 침입예방체계도 불가피하게 실패할수 있다. 체계의 2차방어선은 침입검출이며 이것은 최근년간 많은 연구사업의 초점으로 되여 왔다. 이러한 관심은 다음과 같은 많은 고찰들에 의하여 제기되였다. 즉

1. 침입을 매우 빨리 검출하면 어떤 파괴가 이루어 지기전에 또는 일부 자료가 위태롭게 되기전에 침입자를 식별하고 추방할수 있다. 지어 만일 검출이 침입자를 선취하는데 시간적으로 충분하지 못하다고 하더라도 침입이 빨리 검출될수록 파괴되는 량은 적어 질것이며 회복시간은 보다 빨라 질것이다.
2. 효과적인 침입검출체계는 방해물로서 봉사할수 있으며 따라서 침입을 예방할수 있다.
3. 침입검출은 침입예방기능을 강화하는데 사용할수 있는 침입수법들에 대한 정보를 수집할 가능성을 준다.

침입검출은 침입자들의 행동이 합법적사용자의 행동과 정량화될수 있는 방법상에서 다르다는 가정에 기초하고 있다. 물론 침입자에 의한 공격과 합법적사용자에 의한 자원들에 대한 일반 사용사이에 명백하고 정확한 차이가 있다고 말하기는 어렵다. 차라리 일부 같은것이 있을것이라고 보아야 한다.

그림 15-6에서는 침입검출체계의 설계자들이 직면하고 있는 과제의 특징을 매우 추상적인 말로 보여 주고 있다. 비록 침입자의 전형적인 행동이 합법적사용자의 전형적인 행동과 차이난다고 하더라도 이 행동들에는 겹치는 부분이 있다. 그래서 더 많은 침입자들을 포착할수 있는 침입자행동에 대한 대략적인 해석은 많은 《잘못된 긍정》 즉 인정된 합법적인 사용자들을 침입자라고 판단하는 경우도 발생할수 있다. 다른한편 침입자행동에 대한 정확한 해석에 의하여 잘못된 긍정을 제한하기 위한 시도는 잘못된 부정 즉 침입자를 침입자로 판단하지 않게 하는 오유를 증가시킨다. 따라서 침입검출의 실천에서는 공부정에 대한 전면적고찰과 기교의 요소가 존재한다.

Anderson의 연구[ANDE80]에서는 누구든지 가장한 자와 합법적사용자사이를 적당한 믿음성을 가지고 구분할수 있다는것을 주장하였다. 합법적사용자행동의 양상을 지나간 력사를 고찰하는것으로 확립할수 있으며 그러한 양상들로부터 중요한 잘못된 부정들이 검출될수 있다. Anderson은 직권람용자(권한밖의 방식으로 일하는 합법적사용자)를 검출하는 과제가 비정상적인 행동과 정상적인 행동사이의 차이가 작은것으로 하여 매우 힘들다는것을 지적하였다. Anderson은 그러한 위반들은 오직 이상한 행동에 대한 연구를 통해서는 검출할수 없다고 결론하였다. 그러나 승인되지 않은 사용을 제기하는 조건들의 부류를 명석하게 정의하여 직권람용자의 행동을 검출할수 있다. 마지막으로 비밀사용자에 대한 검출은 순수 자동화된 령역을 넘어 선다고 생각한다. 이 고찰방법들은 1980년대에 론의되였으며 지금도 옳은것으로 하고 있다.

[PORR92]는 침입검출에 대한 다음의 방안들을 정의하고 있다.

1. **통계적위반검출 :** 한주기시간이상 합법적사용자들의 행동에 관계되는 자료를 수집한다. 이 통계적시험들은 행동이 합법적사용자의 행동인가 아닌가 하는 높은 수준의 믿음성을 결정하기 위하여 관측된 행동에 적용된다.

   ㄱ) 턱값검출 : 이 방법은 여러가지 사건들의 빈도에 대하여 사용자와 무관계한 턱값들을 정의하는것을 포함한다.

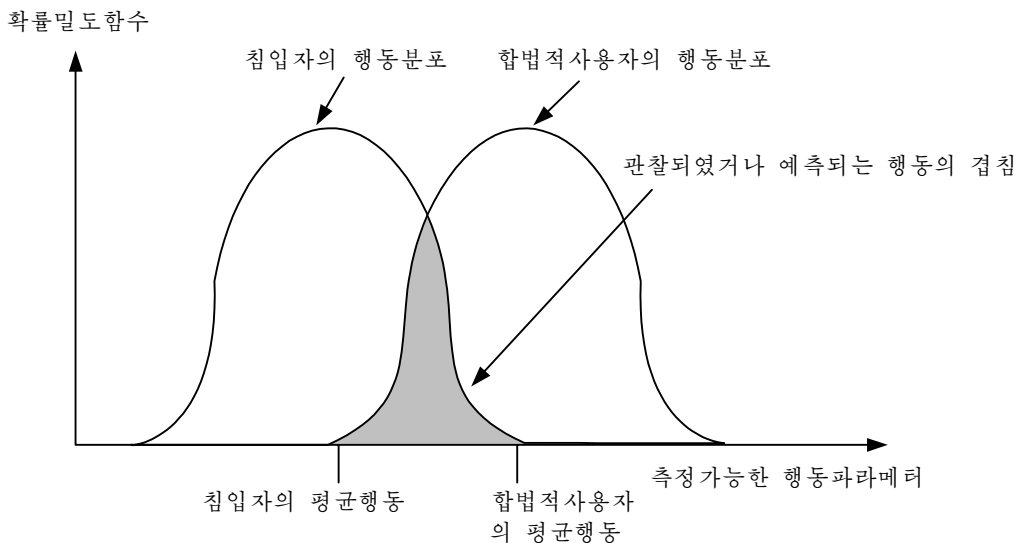



그림 15-6. 침입자와 합법적사용자들의 행동분포

ㄴ) 분포도에 의한 검출 : 매 사용자의 활동분포도는 개별적가입자들의 행동에서의 변화들을 검출하기 위하여 개발되고 사용된다.

2. **규칙에 의한 검출:** 주어 진 행동이 침입자의 행동인가를 결정하기 위하여 사용될수 있는 규칙들의 모임을 정의하기 위한 시도를 포함한다.

   ㄱ) 위반검출 : 이전 사용행동방식으로부터의 리탈을 검출하기 위하여 규칙들이 개발된다.

   ㄴ) 침입식별 : 의심스러운 행동에 대하여 조사하는 전문가체계방법이다.

요약하여 말하면 통계적인 방법들이 정상 즉 기대되는 행동들을 정의하려고 한다면 규칙에 기초한 방법들은 타당한 행동을 정의하려고 한다.

앞에서 설명한 공격자들의 형태에 따르는 통계적위반검출은 그 가입자들의 행동형태를 모의하려고 하는 가장한 자들에 한하여 효과적이다. 한편 그러한 수법들에 의해서는 직권람용자들에 대처할수 없을수 있다. 이러한 공격들에 대해서는 규칙에 기초한 방법들이 문맥상침입을 폭로하는 사건들과 순차들을 알아 차릴수 있다. 실천에서 체계는 넓은 령역의 공격들을 효과적으로 막기 위하여 두 방법들을 결합하여 사용한다.

침입검출을 위한 기본적인 도구는 총화기록이다. 진행된 활동에 대한 기록들은 침입검출체계에 입력하여 사용자의 행동을 보관하고 있어야 한다. 기본적으로 두가지 계획이 사용된다. 즉

- **1차적인 총화기록들 :** 실제로 모든 다중사용자조작체계는 사용자의 동작정보를 수집하는 총화쏘프트웨어를 가지고 있다. 이 정보를 사용하는 유리한 점은 추가적인 수집쏘프트웨어가 요구되지 않는다는것이다. 불리한 점은 1 차총화기록들이 필요한 정보를 포함하지 않을수 있다는것 또는 그것을 편리한 형태로 포함하지 않을수 있다는것이다.
- **검출용검열기록들 :** 수집기능은 침입검출체계가 요구하는 그 정보만을 포함하는 총화기록들을 발생하여 실현할수 있다. 그러한 방법의 한가지 유리한 점은 그것이 제작자에 무관계하게 만들어 질수 있으며 다양한 체계에 맞는다는것이다. 불리한 점은 사실상 한 기계에서 실행하는 두개의 총화기록들을 가지는것으로 초래되는 파잉처리이다.

검출용검열들의 좋은 실례로서 Dorothy Denning[DENN87]에 의하여 개발된것이 있다. 매개 총화기록은 다음의 항목들을 포함한다.

- **주동체 :** 작용들의 시동자. 주동체는 대표적으로 말단사용자이지만 사용자 또는 사용자그룹을 대신하여 작용하는 프로쎄스일수 있다. 모든 동작은 주동체에 의하여 제시된 지령들을 통하여 일어 난다. 주동체들은 서로 다른 접근부류들로 그룹화될수 있으며 이 부류들은 겹쳐 질수 있다.
- **작용 :** 주동체에 의하여 또는 객체와 함께 수행된 동작 례를 들어 가입, 읽기 그리고 입출력수행, 집행 등이다.
- **객체 :** 작용들의 감수기 실례로 파일, 프로그람, 통보문, 기록, 명단, 인쇄기사용자 또는 프로그람으로 창조된 구조들이 속한다. 주동체가 전자우편과 같은 작용의 수용자일 때 그 주동체는 객체로 고찰된다. 객체들은 형에 따라 그룹화될수 있다. 객체의 크기는 객체의 형과 환경에 따라 변할수 있다. 례를 들어 자료기지 작용들은 전체로서 또는 레코드준위에서 자료기지를 총화할수 있다.
- **례외조건 :** 만약 있다면 례외조건은 복귀시에 제기된다.

- **자원사용정형 :** 매개 요소가 어떤 자원을 사용한 량을 보여 주는 량적요소들의 목록(례: 인쇄 또는 연시된 행들의 수, 읽기 또는 쓰기된 레코드들의 수, 처리기 시간, 사용입출력장치, 대화경과시간)
- **시간압인 :** 작용이 발생할 때를 식별하는 유일한 시간날자압인

대부분의 사용자조작들은 많은 요소적인 작용들로 이루어 진다. 실례를 들어 파일복사는 접근타당성검출의 집합과 복사의 설정과 함께 한 파일로부터 읽기 또다른 파일에로 쓰기를 포함하는 사용자지령의 실행을 포괄한다.

현재의 등록부로부터 <Library>등록부에 실행파일 GAME 를 봉사하기 위하여 Smith 가 만든 다음의 지령을 고찰하자.

COPY GAME.EXE TO <LIBRARY> GAME.EXE

다음의 총화기록들이 생겨 날수 있다.

Smith	execute	<Library> COPY.EXE	0	CPU=0002	11058721678
Smith	read	<Smith> GAME.EXE	0	RECORDS=0	11058721679
Smith	execute	<Library>COPY.EXE	Write-viol	RECORDS=0	11058721680

이 경우 복사는 Smith <Library>에로의 쓰기허가를 가지지 않으므로 실패한다. 사용자조작을 요소작용으로 분해하며 세가지 유리한 점을 가진다. 즉

1. 객체들은 체계에서 보호할수 있는 실체들이기때문에 요소작용들을 사용하여 객체에 영향을 주는 모든 행동을 검열하게 한다. 따라서 체계는 접근조종들이 시도한 파괴들을 검출할수 있으며(복귀된 례외조건수의 이상변화를 고려하는 방법으로) 주동체에 접근할수 있는 객체들의 모임안에서의 변화를 고려하여 성과적으로 파괴를 검출할수 있다.
2. 단순객체, 단순작용총화기록들은 모형과 실현을 단순하게 한다.
3. 검출특정총화기록들의 구조가 간단하고 통일적이기때문에 1 차적인 총화기록들로부터 검출목적총화기록들에로 간단히 사영하여 이러한 정보 또는 최소한 일부 정보를 상대적으로 쉽게 얻을수 있다.

제 4 절. 비법적인 쏘프트웨어

아마도 콤퓨터체계들에 대한 가장 정교한 형태의 위협들은 계산체계들에서 취약성들을 사용하는 프로그람들로부터 제기될것이다. 이러한 상황에서는 편집기와 콤파일러와 같은 편의프로그람은 물론 응용프로그람들을 관심하게 된다. 그러한 위협들에 대한 일반용어가 비법적쏘프트웨어 또는 멀웨어(malware)이다. 멀웨어는 목적콤퓨터의 자원들에 대한 파괴를 일으키거나 자원을 소모하기 위하여 설계된 쏘프트웨어이다. 그것은 자주 합법적쏘프트웨어안에 숨겨 지거나 합법적인 쏘프트웨어로 가장한다. 일부 경우에는 그것이 전자우편 또는 감염된 플로피디스크를 통하여 다른 콤퓨터들에 전파될수 있다. 이 절에서는 먼저 그러한 쏘프트웨어위협의 범위를 개괄한다. 이 절의 나머지부분에서는 비루스들을 고찰하는데 먼저 그것의 특징을 보고 그다음에 대책들을 설명한다.

비법적인 프로그람

그림 15-7에서는 비법적쏘프트웨어의 전반적인 분류를 보여 주고 있다. 이 위협들은 두가지 부류로 가를수 있다. 즉 가입자프로그람을 요구하는것들과 요구하지 않는것들이다. 전자는 본질적으로 어떤 실제적인 응용프로그람, 편의프로그람 또는 체계프로그람과 독립적으로 존재할수 없는 프로그람부분들이다. 후자는 조작체계에 의하여 일정작성되고 실행될수 있는 자체포함프로그람들이다.

또한 복제하지 않는 쏘프트웨어위협들과 복제하는 쏘프트웨어위협들의 차이를 구분할수 있다. 전자는 가입자가 특정의 기능을 수행하기 위하여 접근할 때 동작하는 프로그람부분들이다. 후자는 프로그람부분(비루스) 또는 독립프로그람(벌레, 세균)들로 이루어지는데 그것은 실행될 때 같은 체계 또는 어떤 다른 체계상에서 후에 작용하게 되는 한개 또는 그이상의 자기 복제품들을 만들수 있다.

비록 그림 15-7의 분류가 론의하고 있는 정보를 조직화하는데서는 쓸모 있지만 그것은 전부가 아니다. 특히 론리폭탄들 또는 트로이목마들은 비루스 또는 벌레의 한 부분으로 될수도 있다.

함정문

함정문은 그것을 알고 있는 일부 사람들이 보통 보안접근수속들을 거치지 않고 접근권을 얻도록 하는 프로그람의 비밀입구점이다. 함정문들은 프로그람들을 오유수정하고 시험하기 위하여 프로그람작성자들이 오래동안 합법적으로 사용해 왔다. 이것은 보통 프로그람작성자가 인증수속이나 긴 설치시간을 가지는 응용프로그람을 개발하고 있을 때 또는 응용프로그람을 실행하기 위하여 사용자에게 많은 서로 다른 값들을 입력할것을 요구할 때 수행된다. 프로그람을 오유수정하기 위하여 개발자는 특별한 특권을 얻고 싶어하거나 모든 필요한 설치와 인증을 피하기를 원할수도 있다. 프로그람작성자는 또한 응용프로그람안에 설치되여 있는 인증수속에 무엇인가 고장이 있는 프로그람을 활성화시키는 방법이 있다는것을 확신하고 싶어 할수도 있다. 함정문은 입력의 일부 특정한 순차를 인식하는 코드 또는 어떤 사용자ID로부터의 실행에 의해 또는 사건들의 적합하지 않은 순차에 의하여 시동되는 코드이다.

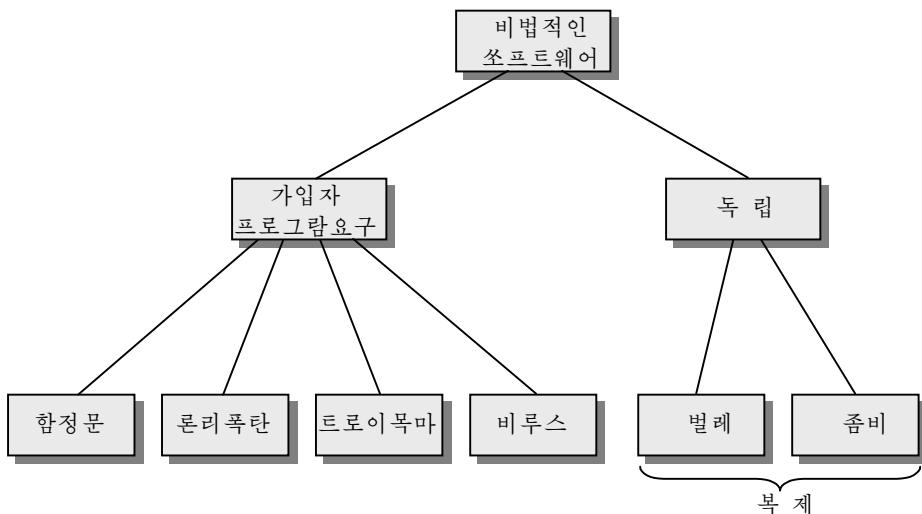



그림 15-7. 비법적프로그람의 분류

함정문들은 비량심적인 프로그람작성자들이 비합법적인 접근을 얻기 위하여 그것들을 사용할 때 위협들로 된다. 함정문은 영화 ≪전쟁게임≫[COOP89]에서 표현된 취약성에 대한 기본개념이였다. 다른 실례는 Multics의 개발기간에 침입시험들이 공군《범팀》에 의하여 진행된것이였다(적수모의). 적용된 한가지 방책은 Multics를 실행하는 싸이트에 위조조작체계갱신판을 보내는것이였다. 갱신판은 트로이목마(후에 서술한다.)를 포함하고 있는데 그것은 함정문에 의하여 활성화될수 있고 범팀이 접근권을 얻을수 있게 한다. 위협은 Multics의 개발자들이 함정문의 존재를 통보받은후에조차 그것을 발견할수 없으리만큼 잘 실현되였다[ENGE80]. 함정문들에 대한 조작체계조종을 실현하기는 힘들다. 보안수단들은 프로그람개발과 쏘프트웨어갱신활동들에 집중하여야 한다.

론리폭탄

프로그람위협의 가장 오랜 형태의 하나는 론리폭탄이다. 론리폭탄은 일정한 조건이 만족되면 《폭발》하도록 설정된 합법적인 프로그람안에 내장된 코드이다. 론리폭탄의 격발기로서 사용될수 있는 조건들의 실례들로서는 어떤 파일들의 존재 또는 결여, 주 또는 날자의 특정한 날 또는 응용프로그람을 실행하는 특별한 사용자를 들수 있다. 한가지 유명한 실례[SPAF89]에서 론리폭탄은 어떤 종업원의 ID번호(폭탄의 제조자의것)를 검사하고 만일 ID가 두번 련속적으로 로임지불계산에 잘못 나타나면 격발되게 되여 있다. 일단 격발되면 폭탄은 자료 또는 전체 파일들을 교체하거나 삭제하고 기계를 정지시키거나 어떤 다른 손상을 줄수 있다. 론리폭탄을 어떻게 적용하는가를 보여 주는 뚜렷한 실례로서Maryland의 Montgomery주의 도서관체계의 경우를 들수 있다[TIME90]. 콤퓨터류통체계를 개발한 계약자는 자기가 돈을 지불 받지 못하면 어떤 날자에 체계를 못쓰게 하는 론리폭탄을 삽입하였다. 체계가 응답시간이 떠서 서고가 자기의 마지막 지불을 보류했을 때 계약자는 론리폭탄의 존재를 알리고 지불을 하지 않으면 그것을 터지게 하겠다고 위협하였다.

트로이목마

트로이목마는 쓸모 있거나 외면상 쓸모 있는 프로그람 또는 지령수속으로써 호출될 때에는 어떤 원하지 않거나 해로운 기능을 수행하는 은페된 코드를 포함하고 있다.

트로이목마프로그람들은 비법적인 사용자가 직접 달성할수 없었던 기능들을 간접적으로 달성하기 위하여 사용될수 있다. 실례를 들어 공유된 체계상에서 다른 사용자의 파일들에 접근하기 위하여 사용자는 실행시 사용자의 파일호출승인을 변경시키는 트로이목마프로그람을 창조하여 임의의 사용자가 파일들을 읽을수 있게 한다. 그다음 제작자는 공통등록부에 그 프로그람을 배치하고 쓸모 있는 편의프로그람이 나타난것처럼 이름을 지어 사용자들이 그것을 실행하도록 유도할수 있다. 한가지 실례로서 사용자의 파일들의 목록을 표면상 희망하는 형태로 생성하는 프로그람이다. 또다른 사용자가 그 프로그람을 실행한후에 제작자는 사용자파일들의 정보에 접근할수 있다. 검출하기 힘든 트로이목마프로그람의 한가지 실례는 체계가입프로그람과 같은 어떤 프로그람들에 그것들이 콤파일될 때 보충적인 코드를 삽입하기 위하여 변경된 콤파일러이다[THOM84]. 그 코드는 트로이목마제작자가 특수한 통과암호를 사용하여 체계에로 가입하게 하는 가입프로그람안에 함정문을 창조한다. 이 트로이목마는 가입프로그람의 원천코드를 읽는것으로는 결코 발견될수 없다.

트로이목마의 또다른 일반목적은 자료파괴이다. 프로그람이 쓸모 있는 기능을 수행하고 있는것처럼 보이지만(실례를 들어 계산프로그람) 그것은 사용자파일들을 완전히 삭제할수 있다. 실례를 들어 CBS집행부는 자기의 콤퓨터기억기안에 포함된 모든 정보를

파괴한 트로이목마에 의하여 못쓰게 된다[TIME90]. 트로이목마는 전자게시판체계에 제공된 도형처리루틴안에 이식되였다.

비루스

비루스는 다른 프로그람들을 변경시켜 그것들을 《감염시킬수》 있는 프로그람이다. 변경판프로그람은 비루스프로그람의 사본을 포함하는데 그 프로그람은 다른 프로그람들을 련속 감염시킬수 있다.

생물학적비루스들은 살아 있는 세포의 조직을 계승할수 있으며 그것을 사용하여 원비루스의 수천개의 완전한 복제품을 만들게 할수 있는 유전코드 DNA 나 RNA 의 작은 토막들이다. 그것의 생물학적사본과 같이 콤퓨터비루스는 자기자체의 완전한 사본을 만들기 위한 수단을 자기의 명령코드안에 가지고 있다. 가입자콤퓨터에 숨어 있는 전형적인 비루스는 콤퓨터의 디스크조작체계를 림시로 조종한다. 그다음 감염된 콤퓨터가 감염되지 않은 쏘프트웨어들과 접촉하게 될 때 비루스의 생신한 사본은 새로운 프로그람안으로 들어 간다. 따라서 감염은 디스크를 교체하거나 망을 통하여 다른 디스크들에 프로그람을 보내는 사용자들의 의심을 받음이 없이 콤퓨터로부터 콤퓨터에로 전파될수 있다. 망환경에서 다른 콤퓨터의 응용프로그람과 체계봉사들을 호출할수 있는 능력은 비루스전파를 위한 배양지를 제공하는것으로 된다.

비루스들은 이 절의 뒤부분에서 더 구체적으로 설명한다.

벌레

망벌레충프로그람들은 체계에서 체계에로 전파하기 위하여 망련결들을 사용한다. 일단 체계내에서 능동상태에 놓이면 벌레는 콤퓨터비루스 또는 세균과 같이 행동할수 있거나 트로이목마프로그람들을 이식하거나 여러가지 분렬 또는 파괴적인 작용들을 할수 있다. 자기자체를 복제하기 위하여 벌레는 몇가지 종류의 망매개물을 사용한다. 실례를 들면 다음과 같은것들이 있다. 즉

- 전자우편기능: 벌레는 자기자체의 사본을 다른 체계에 우편송신한다.
- 원격실행능력: 벌레는 다른 체계상에서 그자체의 사본을 집행한다.
- 원격가입능력: 벌레는 사용자와 같이 원격체계에 가입하고 한 체계로부터 다른 체계에로 그자체를 복제하기 위한 지령들을 사용한다.

웜프로그람의 새로운 사본은 그것이 같은 형식으로 계속 전파되는 원격체계상에서 그 체계에서 수행하는 기능들외의 다른 기능들을 추가하여 실행된다.

망기생충은 콤퓨터비루스와 같은 특징 즉 잠복단계, 전파단계, 시동단계 그리고 집행단계를 나타낸다. 전파단계는 일반적으로 다음과 같은 기능을 수행한다. 즉

1. 감염시키기 위하여 가입자표들 또는 그와 류사한 원격체계주소들의 보관장소들을 조사하는것으로 다른 체계들을 탐색한다.
2. 원격체계와의 련결을 확립한다.
3. 자기자체를 원격체계에 복제하고 그 사본을 실행시킨다.

망벌레는 또한 체계에 자기자체를 복제하기전에 이미 감염되여 있었는가를 결정하려고 할수 있다. 다중프로그람처리체계에서 망벌레는 또한 자기에게 체계프로쎄스와 같은 이름을 달거나 체계조작자가 알수 없는 다른 어떤 이름을 사용하여 자기 존재를 파장할수 있다.

비루스의 경우와 마찬가지로 망벌레들은 알아 내기가 힘들다. 그러나 망보안과 단일체계보안수단들을 적합하게 설계하고 실현한다면 벌레의 위협을 최소로 할수 있다.

좀비

좀비는 다른 인터네트에 접속된 콤퓨터로 비밀리에 퍼지는 프로그람이며 그 콤퓨터를 사용하여 좀비의 생성자를 추적하기 힘든 공격들을 개시한다. 좀비는 대표적으로 Web 싸이트를 겨냥한 봉사거절공격들에 사용한다. 좀비는 의심하지 않는 3 자에 속하는 수천대의 콤퓨터들에 이식되며 인터네트정보흐름에 대한 무자비한 공격을 개시하여 목적 Web 싸이트를 파괴하는데 사용된다.

비루스의 성질

비루스는 다른 프로그람들이 하는 모든것을 할수 있다. 유일한 차이는 비루스가 다른 프로그람에 붙어 가입자프로그람이 실행될 때 몰래 집행하는것이다. 비루스가 일단 집행되면 파일과 프로그람의 삭제와 같은 임의의 기능을 수행할수 있다.

비루스의 생존기간에 전형적인 비루스들은 다음의 4개 단계를 거친다. 즉

- **잠복단계** : 비루스는 놀고 있다. 비루스는 날자, 다른 프로그람이나 파일의 존재 또는 디스크용량의 어떤 한계의 초과와 같은 어떤 사건에 의하여 우발적으로 활성화된다. 모든 비루스가 이 단계를 거치는것은 아니다.
- **전파단계** : 비루스는 자기와 동일한 사본을 다른 프로그람 또는 디스크의 일정한 체계령역에 배치한다. 감염된 매개 프로그람은 그자체가 전파단계에 들어 갈 비루스복제품을 포함하게 된다.
- **시동단계** : 비루스는 능동상태로 되여 목적하는 기능을 수행한다. 잠복단계의 경우와 마찬가지로 시동단계는 비루스사본이 몇번 복제되였는가 하는 회수를 비롯하여 여러가지 체계사건들에 의하여 일어 날수 있다.
- **집행단계** : 기능이 수행된다. 기능은 화면에 통보문의 표시와 같이 아무런 손상도 주지 않는것일수도 있으며 프로그람과 자료파일의 파괴와 같이 손상을 주는것일수도 있다.

대부분의 비루스들은 특정한 조작체계에 특유한 방식으로 어떤 경우에는 특정한 하드웨어기동환경에 특유한 방식으로 자기 작업을 수행한다. 따라서 그것들은 특정한 체계의 세부들과 약점들을 사용할수 있도록 설계된다.

비루스의 류형

비루스가 처음 출현한 때로부터 비루스제작자와 항비루스쏘프트웨어제작자사이에 끊임 없는 대결이 진행되여 왔다. 현존하는 비루스형태들에 대한 효과적인 대책들이 개발됨에 따라 새로운 형태의 비루스들이 개발되여 왔다. [STEP93]에서는 가장 중요한 비루스형태에 속하는 다음과 같은 부류들을 제안하고 있다. 즉

- **기생비루스 :** 전통적이며 현재 가장 일반적인 비루스형태. 기생비루스는 감염된 비루스가 집행될 때 감염시킬 다른 집행가능한 파일들을 발견하여 자기자신을 집행파일들에 붙여서 복제한다.
- **기억기상주비루스 :** 상주체계프로그람의 부분으로서 주기억기에 있는 작은 구역. 주기억기에 상주한 시점에서부터 비루스는 집행하는 매개 프로그람을 감염시킨다.
- **기동분구비루스 :** 체계가 비루스를 포함하는 디스크로 기동될 때 마스터기동레코드 또는 기동레코드를 감염시키고 전파시킨다.

- **스텔스비루스 :** 항비루스쏘프트웨어에 의한 검출로부터 자기자신을 숨기기 위하여 충분히 설계된 비루스형태
- **다형성비루스 :** 매번 감염될 때마다 변화되여 비루스의 《서명》에 의한 검출을 불가능하게 하는 비루스

스텔스비루스의 한가지 실례로 감염된 프로그람이 감염되지 않은 프로그람과 똑같은 길이가 되도록 압축을 사용하는 비루스를 들수 있다. 보다 더 세련된 수법들이 있을수 있다. 실례로 비루스는 입출구루틴들을 사용하여 디스크의 의심되는 부분들을 읽으려 할 때 감염되지 않은 원래의 프로그람으로 다시 출현하도록 이 루틴들에 차단론리를 배치할 수 있다. 따라서 스텔스는 이와 같이 비루스에 적용하는 용어가 아니라 비루스가 검출을 회피하기 위하여 사용하는 수법을 가리키는 용어이다.

다형성비루스는 기능적으로는 동등하지만 명백히 다른 비트패턴들을 가지는 복제가 진행되는동안 복제품들을 창조한다. 스텔스비루스와 마찬가지로 목적은 비루스들을 주사하는 프로그람들을 좌절시키는것이다. 이 경우에 비루스의 《서명》은 매개 복제품에 따라 변하게 된다. 이러한 변화를 일으키기 위하여 비루스는 임의로 불필요한 명령들을 삽입하거나 독립적인 명령들의 순서를 교체할수 있다. 더 효과적인 방법은 암호화를 사용하는것이다. 일반적으로 변이엔진이라고 하는 비루스의 부분은 임의의 암호화열쇠를 생성하여 비루스의 나머지부분을 암호화한다. 열쇠는 비루스와 함께 기억되며 변이엔진 그 자체는 변화된다. 감염된 프로그람이 호출되면 비루스는 기억된 임의의 열쇠를 사용하여 비루스를 해독한다. 비루스는 복제할 때 서로 다른 임의의 열쇠를 선택한다.

비루스제작자들의 병기고에 있는 또다른 무기는 비루스생성도구틀이다. 이러한 도구틀은 비교적 초학자가 여러개의 서로 다른 비루스들을 빨리 생성하도록 한다. 도구틀에 의하여 생성된 비루스들이 비록 무에서 설계된 비루스들보다 세련되지 못할수 있지만 생성되는 몇개 안되는 새로운 비루스들은 항비루스법들에 문제를 일으킨다.

또다른 비루스제작자의 도구는 비루스교체게시판이다. 여러개의 이러한 게시판들이 미국과 다른 나라들에서 나타나기 시작하였다[ADAM92]. 이 게시판들은 비루스들의 생성을 위한 정보는 물론 내리적재될수 있는 비루스들의 사본들을 제시한다.

마크로비루스

최근년간에 회사의 싸이트들에서 맞다 드는 비루스의 수는 눈부시게 증가하고 있다. 실제로 이 증가는 모두 가장 새로운 비루스류형들중의 하나인 마크로비루스의 증식에 인한것이다. 1999년 후반기현재 마크로비루스는 모든 콤퓨터비루스의 3분의 2를 차지하였다[KABA99].

마크로비루스는 특히 몇가지 원인으로 하여 위협으로 되고 있다. 즉

1. 마크로비루스는 가동환경에 무관계하다. 실제로 모든 마크로비루스들은 Microsoft Word문서들을 감염시킨다. Word를 지원하는 모든 하드웨어가동환경과 조작체계는 감염될수 있다.
2. 마크로비루스는 집행가능한 코드부분들이 아니라 문서들을 감염시킨다. 콤퓨터체계에 있는 대부분의 정보는 프로그람형식보다 문서형식으로 존재한다.
3. 마크로비루스는 쉽게 전파된다. 매우 공통적인 전파방법은 전자우편에 의한 전파이다.

마크로비루스들은 Word와 Microsoft Excel과 같은 다른 사무응용들에서 보게 되는 특징 즉 마크로를 사용한다. 마크로는 본질상 Word처리문서나 다른 류형의 파일과 조

합되는 집행가능한 프로그람이다. 대표적으로 사용자들은 마크로들을 사용하여 반복되는 과제들을 자동화하고 그것에 의해서 건눌림상태들을 보관한다. 마크로언어는 보통 베이씨크프로그람작성언어의 형태와 같다. 사용자는 건눌림순서를 마크로로 정의하고 기능건 또는 건들의 특수한 짧은 조합이 입력될 때 마크로가 호출되도록 설정할수 있다.

마크로비루스를 창조하게 하는것은 자동집행마크로이다. 이것은 명백한 사용자입력이 없이 자동적으로 호출되는 마크로이다. 일반적인 자동집행사건들은 파일열기, 파일닫기, 응용프로그람의 시동이다. 마크로는 일단 실행하면 자기자신을 다른 문서들에 복사하고 파일들을 삭제하며 사용자체계에 여러가지 손상을 준다. Microsoft Word에는 세가지 종류의 자동집행마크로들이 있다. 즉

- **자동집행 :** AutoExec라는 이름을 가진 마크로가 Word의 시동등록부에 보관되여 있는 《normal.dot》본보기 또는 전역본보기에 있으면 그것은 Word가 기동될 때마다 집행된다.
- **자동마크로 :** 자동마크로는 문서의 열기 또는 닫기, 새 문서의 창조 또는 Word의 탈퇴와 같은 규정된 사건이 일어날 때 집행된다.
- **지령마크로 :** 전역마크로파일 또는 문서에 련결된 마크로가 현존 Word지령의 이름을 가지면 사용자가 그 지령(실례로 파일보관)을 호출할 때마다 지령마크로가 집행된다.

마크로비루스를 전파시키는 공통적인 수법은 다음과 같다. 자동마크로 또는 지령마크로는 전자우편 또는 디스크이송으로 체계에 넣어 진 Word문서에 련결된다. 문서가 열린 후에 어떤 시점에서 마크로가 집행된다. 마크로는 전역마크로파일에 자기자체를 복사한다. 다음번 Word와의 대화를 시작하면 감염된 전역마크로가 동작한다. 이 마크로가 집행되면 그것은 자기를 복제할수 있으며 손상을 줄수 있다.

Word의 련속적인 공개판들은 마크로비루스에 대한 보호기능을 확장하고 있다. 실례로 Microsoft는 의심스러운 Word파일들을 검출하며 구매자에게 마크로들을 가지고 있는 파일을 열 때의 잠재적인 위험을 경보하는 선택적인 마크로비루스보호도구를 제공한다. 여러 항비루스제품제작자들도 역시 마크로비루스를 검출하고 수정하는 도구들을 개발하여 왔다. 다른 비루스들과 마찬가지로 마크로비루스분야에서도 경쟁이 계속 확대되고 있다.

항비루스법

비루스의 위협에 대처하기 위한 리상적인 방법은 예방이다. 즉 비루스가 무엇보다 먼저 체계에 들어 오지 못하게 하는것이다. 예방은 비록 성공적인 비루스공격회수를 줄일수 있지만 이 목적을 달성하기는 일반적으로 힘들다. 가장 좋은 방법은 다음과 같은 기능들을 수행하는것이다. 즉

- **검출 :** 일단 감염이 발생하였으면 발생했다는것을 확정하고 비루스위치를 정한다.
- **식별 :** 일단 비루스가 검출되면 프로그람을 감염시킨 특정의 비루스를 식별한다.
- **제거 :** 특정의 비루스가 식별되였으면 감염프로그람으로부터 비루스의 모든 흔적들을 제거하고 그것을 원래상태로 복원한다. 병이 더이상 전파할수 없도록 모든 감염된 체계들에서 비루스를 제거한다.

검출은 성공하였지만 식별이나 제거가 불가능한 경우 대책은 감염된 프로그람을 없애고 깨끗한 여벌판본을 재적재하는것이다.

비루스와 항비루스들은 서로 협조하면서 발전하고 있다. 초기비루스들은 상대적으로

간단한 코드부분들이였으며 상대적으로 간단한 항비루스쏘프트웨어제품들에 의하여 식별되고 제거될수 있었다. 비루스대결이 발전하는데 따라 비루스들과 물론 항비루스쏘프트웨어는 모두 보다 더 복잡해 지고 세련되여 왔다. 점차 세련된 항비루스법들과 제품들이 출현하기 시작하였다. 이 소절에서는 가장 중요한 두가지 방법을 강조한다.

범용암호해제

범용암호해제(GD)수법에 의하여 항비루스프로그람은 빠른 주사속도를 유지하면서 가장 복잡한 다형성비루스들까지도 쉽게 검출할수 있다[NACH97]. 다형성비루스를 포함하고 있는 파일이 집행될 때 비루스는 동작하기 위하여 자기자신을 해독하여야 한다는 것을 상기하자. 이러한 구조를 검출하기 위하여 다음과 같은 요소들을 포함하는 GD주사프로그람에 집행가능한 파일들을 통과시킨다.

- **CPU모방기 :** 쏘프트웨어에 기초한 가상콤퓨터. 집행가능한 파일의 명령들은 밑준위의 처리기에서 집행되는것이 아니라 모방기에서 해석된다. 모방기는 모든 등록기들과 다른 처리기하드웨어에 대한 쏘프트웨어판본들을 포함함으로써 밑준위의 처리기가 모방기에서 해석된 프로그람의 영향을 받지 않도록 한다.
- **비루스서명주사기 :** 알려 진 비루스서명들을 찾는 목적코드를 주사하는 모듈
- **모방조종모듈 :** 목적코드의 집행을 조종한다.

모의를 시작할 때마다 모방기는 목적코드에 있는 명령들을 한번에 한개씩 해석하기 시작한다. 따라서 목적코드가 비루스를 해제하고 적발하는 암호해제루틴을 포함한다면 그 코드는 해석된다. 사실상 비루스는 비루스를 적발하는것으로 항비루스프로그람과 같은 일을 수행한다. 조종모듈은 주기적으로 해석을 중단하고 목적코드에서 비루스서명들을 주사한다.

해석이 진행되는 동안 목적코드는 실제의 개인용콤퓨터환경에 아무런 손상도 주지 않을수 있다. 왜냐하면 그것은 완전히 조종된 환경에서 해석되고 있기때문이다.

GD주사기설계에서 가장 어려운 문제는 매개 해석을 얼마나 오래 실행하는가를 결정하는것이다. 대표적으로 비루스요소들은 프로그람이 집행을 시작한후에 곧 동작하지만 다 그렇지는 않다. 주사기가 특정한 프로그람을 더 오래 모방하면 할수록 그것은 그 어떤 숨은 비루스들도 더 잘 잡아 낸다. 그러나 항비루스프로그람은 사용자가 불평하기전에 제한된 시간과 자원들만을 차지할수 있다.

수자면역체계

수자면역체계는 IBM에 의하여 개발된 비루스보호를 위한 포괄적인 방법이다[KEPH97a, KEPH97b]. 이 체계의 개발동기는 인터네트에 기초한 비루스전파의 위협이 증가된데 있다. 먼저 이 위협들에 대하여 서술하고 IBM의 방법들을 개괄한다.

현재 비루스위협은 새로운 비루스들과 새로운 변종들이 상대적으로 느리게 전파되는 특징을 가지고 있다. 항비루스쏘프트웨어는 대표적으로 달마다 갱신되고 있으며 이것은 비루스문제를 조종하기에 충분하였다. 현재 인터네트는 비루스들의 전파에서 비교적 작은 역할을 놀고 있다. 그러나 [CHES97]이 지적하는바와 같이 인터네트에서 두가지 주요경향은 앞으로 비루스전파속도를 증가시키는데 큰 영향을 줄것이다. 즉

- **통합우편체계 :** Lotus Notes와 Microsoft Outlook와 같은 체계들은 매우 간단히 누군가에게 무엇인가를 송신하거나 수신되는 객체를 연구하도록 한다.
- **이동프로그람체계 :** Java와 Active X와 같은 능력으로 하여 프로그람들은 한

체계에서 다른 체계에로 자기 소유권을 옮길수 있다.

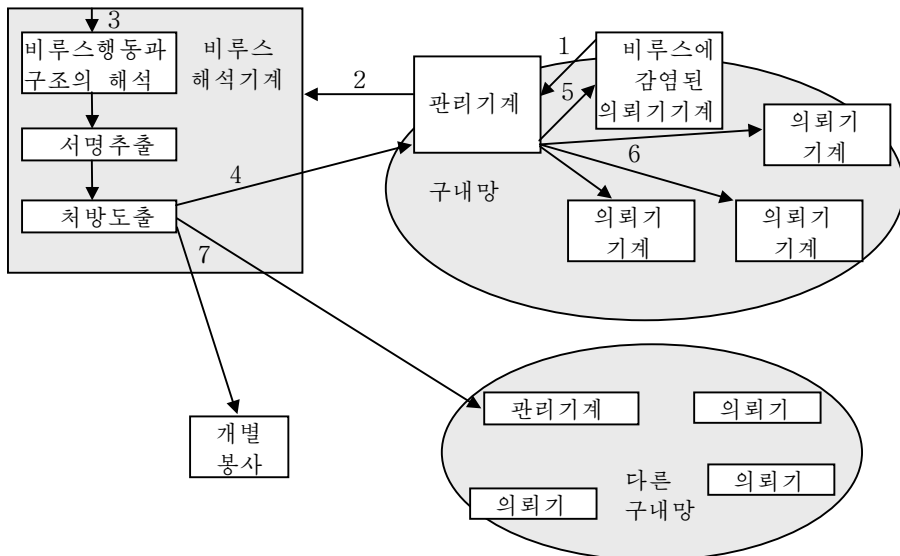
이러한 인터네트에 기초한 능력들로 제기된 위협들에 대응하여 IBM은 표준수자면역체계를 개발하였다. 이 체계는 앞의 소절에서 론의한 프로그람모방의 사용을 발전시키고 범용모방 및 비루스검출체계를 제공한다. 이 체계의 목적은 비루스들이 들어 오자마자 거의 박멸할수 있도록 빠른 응답시간을 제공하는것이다. 새로운 비루스가 기관내에 들어오면 면역체계는 비루스를 포착, 해석, 검출 및 차페하고 제거한다. 그리고 비루스가 다른 곳에서 실행되기전에 검출할수 있도록 IBM Anti Virus를 실행하고 있는 체계에 그 비루스에 대한 정보를 넘겨 준다.

그림 15-8은 수자면역체계조작의 대표적인 단계들을 설명한다.

1. 매개 PC의 감시프로그람은 체계행동, 프로그람의 의심스러운 변경 또는 계렬서명에 기초한 여러가지 경험적방법을 사용하여 비루스가 존재할수 있다는것을 추리한다. 감시프로그람은 감염되였다고 생각되는 프로그람의 사본을 기관내에 있는 관리기계에 전송한다.
2. 관리기계는 표본을 암호화하고 그것을 중앙비루스해석기계에 송신한다.
3. 이 기계는 감염된 프로그람을 해석하기 위하여 안전하게 실행할수 있는 환경을 창조한다. 이러한 목적에 사용되는 수법들은 의심되는 프로그람을 집행하고 감시하는 보호된 환경을 모방하고 생성하는 기능을 가지고 있다. 그다음 비루스해석기계는 비루스를 식별하고 제거하기 위한 처방을 내린다.
4. 결과적으로 얻은 처방은 관리기계에 다시 송신된다.
5. 관리기계는 감염된 의뢰기에 처방을 전송한다.
6. 처방은 역시 기관내의 다른 의뢰기들에 전송된다.
7. 세계의 여기저기에 있는 가입자들은 새로운 비루스들로부터 자기들을 보호하는 정기적인 항비루스갱신판들을 받는다.

수자면역체계의 성공은 새롭고 혁신적인 비루스변종들을 검출할 비루스해석기계의 능력에 의거한다. 빈터에서 발견되는 비루스들을 항상 분석하고 감시함으로써 위협을 물리치기 위한 수자면역체계를 끊임없이 갱신할수 있다.

그림 15-8. 수자면역체계

전자우편비루스

비법적쏘프트웨어에서 가장 최근에 개발된것은 전자우편비루스이다. Melissa와 같은 가장 빨리 전파하는 전자우편비루스는 부속물로 매몰된 Microsoft Word마크로를 사용하였다. 접수자가 전자우편부속물을 열면 Word마크로가 동작한다. 그때

1. 전자우편비루스는 사용자의 전자우편제품에 있는 우편목록의 모든 대상에 자기자신을 송신한다.
2. 비루스는 국부적인 손상을 준다.

1999년 말에 더 강력한 전자우편비루스판본이 나왔다. 이 새로운 판본은 부속물을 여는것으로가 아니라 비루스를 포함하는 전자우편을 여는것으로 드물게 활성화시킬수 있다. 비루스는 전자우편제품이 지원하는 Visual Basic서술언어를 사용한다.

따라서 전자우편을 통하여 도착하고 전자우편쏘프트웨어의 특징을 사용하여 인터네트에 자기자신을 복제하는 멀웨어의 새 세대를 보게 된다. 비루스는 활성화되자마자(전자우편부속물의 열기나 전자우편의 열기로) 자기자신을 감염된 가입자가 알고 있는 모든 전자우편주소들에 전파시킨다. 결과 몇달 또는 몇년에 걸쳐 전파하던 비루스들이 지금은 몇시간내에 전파된다(표 15-4). 이로 하여 많은 손해를 입기전에 항비루스쏘프트웨어가 대응하기는 매우 힘들다. 결국 더 높은 수준의 보안을 PC이상의 망편의프로그람과 응용쏘프트웨어에 구축하여 성장하는 위협에 대처하여야 한다[SCHN99].

표 15-4. 비루스전파시간

비루스	발생한 년	류형	가장 우세하다고 보아 진 기간	판정된 손해
Jerusalem, Cascade, Form	1990	.exe파일	3년	5년이상 모든 비루스에 대하여 5천만딸라
Concept	1995	Word마크로	4달	5천만딸라
Melissa	1999	Word마크로가 허가된 전자우편	4일	3억8천5백만딸라까지
Love letter	2000	VBS에 기초하여 허가된 전자우편	5시간	15조딸라까지

원천: www.icsa.net

제 5 절. 믿음직한 체계

지금까지 론의한것들중에서 대부분은 주어 진 사용자에 의한 피동적인 또는 능동적인 공격으로부터 주어 진 통보문이나 항목을 보호하는것과 관련된것이였다. 어느정도 차이나지만 널리 응용할수 있는 요구는 보안수준에 기초하여 자료 또는 자원들을 보호하는것이다. 이것은 일반적으로 군대에서 실시하고 있는데 거기서는 여러개의 정보를 비밀이 아닌것(U), 덜 비밀인것(C), 비밀인것(S), 극비밀인것(TS) 또는 그이상의것으로 분류하고 있다. 이 개념은 정보를 대략적인 부류들로 조직할수 있고 사용자들이 자료의 일정한 부류들에 접근할수 있도록 허가증을 줄수 있는 다른 령역에서 똑같이 응용할수 있다. 실례로 가장 높은 수준의 보안은 회사의 사무원들과 그들의 책임자만이 접근할수 있는 전략적인 회사계획문서들과 자료에 대하여 실시할수 있으며 다음으로 높은 수준의 보안

은 관리부서사람들과 회사사무원들 기타 등등만이 접근할수 있는 매우 중요한 재정 및 개인자료에 대하여 실시할수 있다.

여러 종류 또는 여러 수준의 자료를 정의할 때의 보안을 **여러준위보안**이라고 한다. 여러준위보안에 대한 요구를 일반적으로 서술한것을 보면 그 흐름이 허용된 사용자의 의사를 반영하지 않는다면 높은 수준에 있는 주동체는 보다 낮은 수준 또는 비교할수 없는 수준에 있는 주동체에로 정보를 전달할수 없다는것이다. 실현을 목적으로 이 요구를 두 부분으로 나누어 간단히 설명한다. 여러준위보안체계는 다음과 같은 대책을 취하여야 한다. 즉

- **읽기금지 :** 주동체는 자기보다 낮은 또는 같은 보안수준의 객체만을 읽을수 있다. 론문에서는 이것을 **단순보안속성**이라고 부르고 있다.
- **쓰기금지 :** 주동체는 자기보다 높거나 같은 보안수준에만 쓰기할수 있다. 론문에서는 이것을 ***-속성**[1](별속성이라고 한다.)이라고 한다.

이 두가지 규칙을 적당히 지키면 여러준위보안이 보장된다. 자료처리체계에서 지금까지 써 오며 많은 연구와 개발의 대상이였던 방법은 참조감시기개념에 기초하고 있다. 이 방법을 그림 15-9에서 설명한다. 참조감시기는 주동체와 객체의 보안파라메터들에 기초하여 객체들에 대한 주동체들의 접근을 조절하는 콤퓨터하드웨어와 조작체계의 조종요소이다. 참조감시기는 매개 주동체가 접근특권들(보안허가증)과 매개 객체에 대한 보호속성들(분류준위)을 목록화한 보안핵심부자료기지라는 파일에 접근한다. 참조감시기는 보안규칙들(읽기금지 ,쓰기금지)을 실시하며 다음의 속성을 가진다. 즉

- **완전중재 :** 보안규칙은 실례로 파일을 열 때가 아니라 파일에 대한 접근 때마다 실시된다.

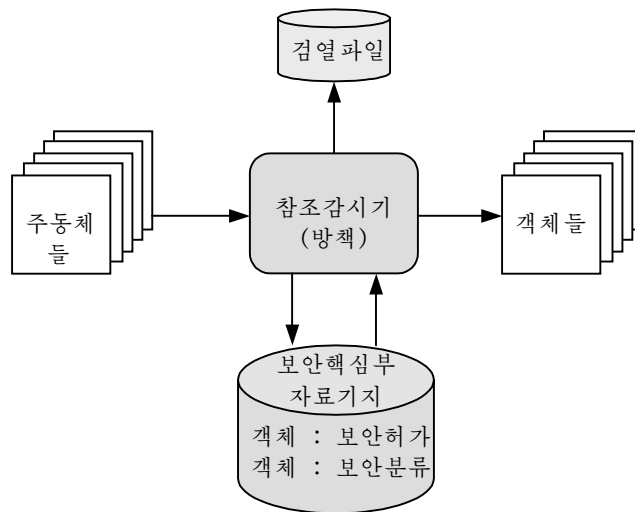



그림 15-9. 참조감시기개념

---

1. 《*》은 아무런 의미도 없다. 모형에 대한 첫 보고서를 작성하는 동안 속성의 적당한 이름을 생각할수 있는 사람은 전혀 없다. 별표는 본문편집기가 속성의 이름이 일단 만들어 지면 그 이름을 사용하는 모든 구체례들을 빨리 찾아 교체할수 있도록 초안에서 입구된 가상문자였다. 이름이 전혀 제시되지 않았으므로 보고서는 《*》을 그대로 둔채 공개되였다.

- **고립 :** 참조감시기와 자료기지는 승인되지 않는 변경으로부터 보호된다.
- **검증성 :** 참조감시기의 정확성은 증명할수 있어야 한다. 즉 참조감시기가 보안규칙들을 실시하며 완전중재와 고립을 제공한다는것을 수학적으로 증명할수 있어야 한다.

이것들은 실현하기 힘든 요구들이다. 완전중재에 대한 요구는 주기억기와 디스크 및 테프에 있는 자료에 대한 매개 접근을 중재하여야 한다는것을 의미한다. 순수 쏘프트웨어적인 실현은 너무 큰 성능악화를 가져 오기때문에 현실적이지 못하다. 해결방도는 일부를 하드웨어로 실현하여야 한다는것이다. 고립에 대한 요구는 공격자가 아무리 솜씨가 있다고 하더라도 참조감시기의 론리나 보안핵심부자료기지의 내용들을 변경할수 없어야 한다는것을 의미한다. 마지막으로 수학적증명에 대한 요구는 범용콤퓨터만큼 복잡하므로 실현하기 힘들다. 이러한 검증을 할수 있는 체계를 **믿음직한 체계**라고 한다.

그림 15-9에서 지적한 마지막요소는 검열파일이다. 보안핵심부자료기지에서 검출된 보안위반들 및 허용된 변경들과 같은 중요한 보안사건들은 검열파일에 기억된다.

미국방성은 1981년에 자기자신의 요구들을 만족시키고 사회적봉사를 위하여 국가안전보장국(NSA)안에 믿음직한 콤퓨터체계들의 광범한 사용을 조장하기 위한 목적으로 콤퓨터쎈터를 설립하였다. 이 목적은 쎈터의 상업제품평가프로그람에 의하여 실현된다. 본질상 쎈터는 대략적인 보안요구들을 만족시키는 상업적으로 사용가능한 제품들을 평가하려고 한다. 쎈터는 평가된 제품들이 보장하는 보안특징들의 범위에 따라 그것들을 분류한다. 이러한 평가들은 국방성의 상품조달을 위하여 필요하며 공개되여 자유롭게 사용할수 있다. 그러므로 그 평가자료는 상업적구매자들이 상업적으로 사용할수 있는 기성장비를 구입하는데서 지침으로 봉사할수 있다.

트로이목마방어

트로이목마공격들을 막기 위한 한가지 방법은 안전하고 믿음직한 신용조작체계를 사용하는것이다. 그림 15-10에서는 한가지 실례를 설명하고 있다[BOEB85]. 이 경우에 트로이목마는 대부분의 파일관리 및 조작체계들이 사용하는 표준보안기구 즉 접근조종목록을 속이는데 사용된다. 이 실례에서 보브라는 사용자는 극히 중요한 문자렬《CPE170KS》을 포함하는 자료파일과 어떤 프로그람을 통하여 대화한다. 사용자 보브는 자기자신을 위하여 집행하는 프로그람들에만 읽기/쓰기허가된 파일을 창조하였다. 즉 보브가 소유한 프로쎄스들만 파일에 접근할수 있다.

트로이목마공격은 알리스라는 적대적인 사용자가 체계에 대한 합법적인 접근권을 얻고 트로이목마프로그람과 《뒤주머니》파일과 같은 공격에 사용되는 개인파일을 다 설치하면 드디여 시작된다. 알리스는 이 뒤주머니파일에 대한 읽기/쓰기허가권을 가지며 보브에게는 쓰기허가권만을 준다(그림 15-10 ㄱ). 알리스는 트로이목마프로그람을 쓸모 있는 편의프로그람인것처럼 선전하여 보브가 그 프로그람을 호출하도록 유도한다. 트로이목마프로그람은 자기가 보브에 의하여 집행되고 있다는것을 검출하면 보브의 파일에서 중요한 문자렬을 읽고 그것을 알리스의 뒤주머니파일에 복사한다(그림 15-10 ㄴ). 읽기 및 쓰기조작들은 다 접근조종목록들에 의해 부과되는 제한조건들을 만족시킨다. 그다음 알리스는 그 문자렬을 알아 내기 위해 오직 보브의 파일에만 접근해야 한다.

이 대본에서 안전한 조작체계를 사용하는 방법을 고찰하자(그림 15-10 ㄷ). 보안준위들은 콤퓨터에 접근중에 있는 말단과 통과암호/ID로 식별된 관련사용자와 같은 특징들에 기초하여 가입시에 주동체들에 할당된다. 이 실례에는 두가지 보안준위 즉 기밀에 속하는것(회색)과 공개하는것(흰색)이 있으며 기밀에 속하는것은 공개하는것보다 더 높

은 준위에 놓이도록 배렬한다. 보브가 트로이목마프로그람을 기동하면(그림 15-10 ㄹ) 그 프로그람은 보브의 보안준위를 획득한다. 따라서 단순보안속성하에서는 중요한 문자렬을 관측할수 있다. 그러나 그 프로그람이 문자렬을 공개파일(뒤주머니파일)에 기억시키려고 할 때 *－속성은 위반되며 기억시도는 참조감시기에 의하여 금지된다. 따라서 뒤주머니파일에로의 쓰기시도는 접근조종목록이 쓰기를 허용한다고 해도 거절된다. 즉 보안방책은 접근조종목록기구보다 우월하다.

제 6 절. Windows 2000의 보안

지금까지 론의하여 온 접근조종개념들의 좋은 실례로 객체지향개념들을 사용하여 강력하고 유연한 접근조종능력을 주는 Windows 2000(W2K)접근조종기능을 들수 있다.

W2K는 프로쎄스, 스레드, 파일, 창문 및 다른 객체들에 적용할수 있는 통일적인 접근조종기능을 제공하고 있다. 접근조종은 두개의 실체 즉 매개 프로쎄스와 관련된 접근통표와 프로쎄스간 접근이 가능한 매개 객체와 관련된 보안서술자에 의하여 결정된다.

접근조종기구

사용자가 W2K체계에 가입할 때 W2K는 이름/통과암호를 사용하여 사용자를 인증한다. 가입이 접수되면 프로쎄스가 그 사용자를 위하여 창조되며 접근통표는 프로쎄스객체와 관련된다. 후에 상세히 서술하는 접근통표는 보안을 위하여 체계가 사용자를 알수 있도록 하는 식별자인 보안ID(SID)를 포함한다. 임의의 추가적인 프로쎄스들이 초기사용자프로쎄스에 의하여 대량적으로 만들어 질 때 새로운 프로쎄스객체는 같은 접근통표를 물려 받는다.

접근통표는 두가지 목적에 봉사한다. 즉

1. 그것은 접근확인속도를 높이기 위하여 필요한 모든 보안정보를 함께 유지한다. 사용자와 관련된 임의의 프로쎄스가 접근을 시도할 때 보안보조체계는 프로쎄스와 관련된 통표를 사용하여 사용자의 접근특권들을 결정할수 있다.
2. 그것은 매개 프로쎄스가 자기를 위하여 실행하고 있는 다른 프로쎄스들에 영향을 주지 않고 제한된 방법으로 자기의 보안특성지표들을 변경시킬수 있다.

두번째 목적의 기본의미는 사용자와 관련될수 있는 특권들과 관련된다. 접근통표는 사용자가 어떤 특권들을 가질수 있는가를 지적한다. 일반적으로 통표는 매개 특권들이 금지된 상태로 초기화된다. 그후에 사용자의 프로쎄스중 한개 프로쎄스가 특권조작을 수행하려고 한다면 프로쎄스는 적당한 특권을 허락하고 접근할수 있다. 하나의 체계규모의 장소에 사용자에 대한 모든 보안정보를 유지하는것은 기대할수 없다. 왜냐하면 그 경우에 한개 프로쎄스에 특권을 허락한다면 모든 프로쎄스들에 특권을 허락하기때문이다.
프로쎄스간 접근이 가능한 매개 객체와 관련되는것은 보안서술자이다. 보안서술자의 주요구성성분은 객체의 여러 사용자들과 사용자그룹들에 대한 접근권들을 규정하는 접근조종목록이다. 프로쎄스가 객체에 접근하려고 한다면 프로쎄스의 SID는 객체의 접근조종목록과 비교하여 접근이 허용되는가를 결정한다.

응용프로그람이 보증할수 있는 객체에로의 참조를 연다면 W2K는 객체의 보안서술자가 응용프로그람의 사용자접근을 허락하는가를 확인한다. 검사가 성공하면 W2K는 허락된 접근권이 허락된 결과객체들을 고속완충한다.

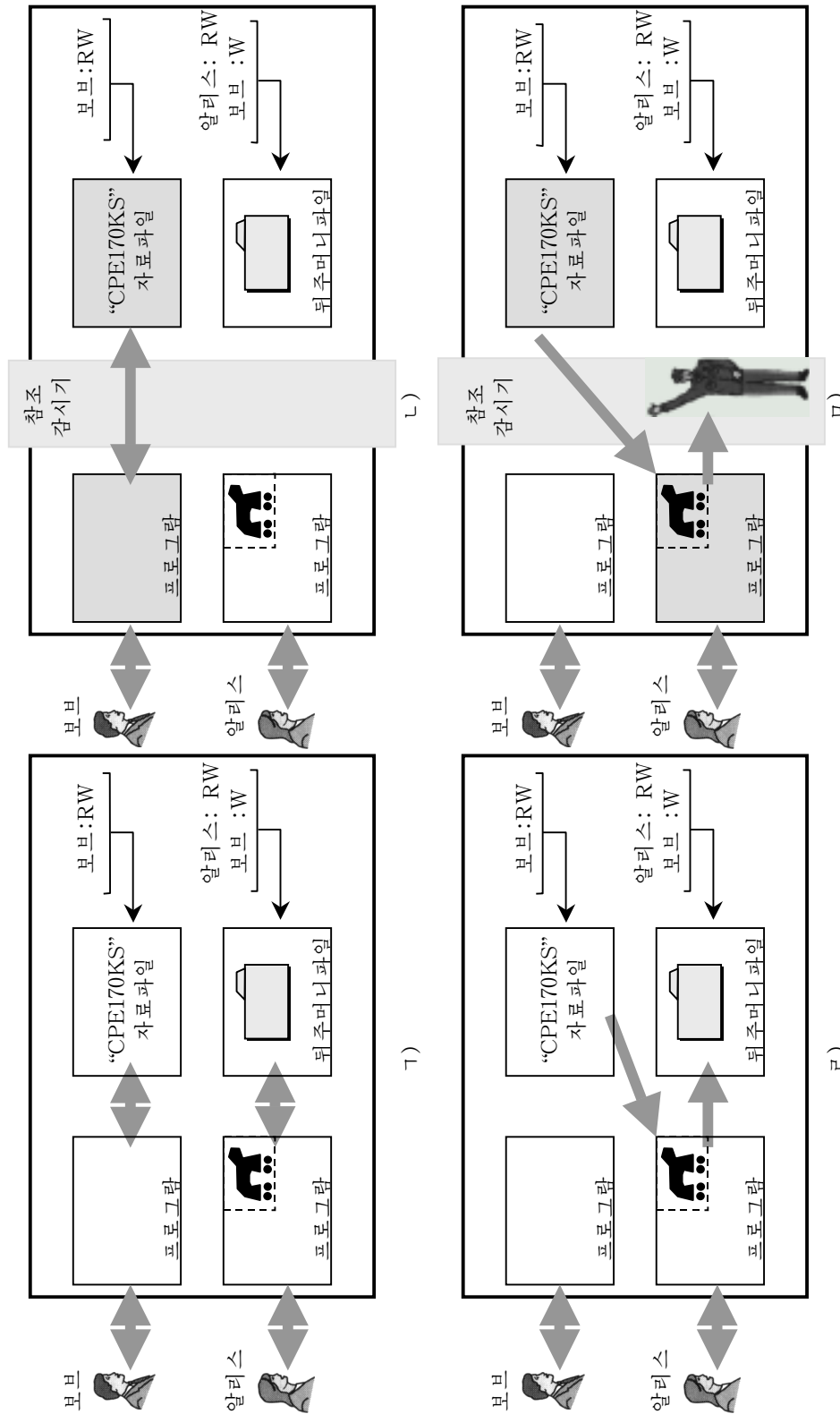



그림 15-10. 트로이목마와 안전한 조작체계

W2K보안의 중요한 특징은 의뢰기/봉사기환경에서 보안의 사용을 단순화하는 의인화개념이다. 의뢰기와 봉사기가 RPC련결을 통하여 대화한다면 봉사기는 의뢰기의 권한들과 관련된 접근에 대한 요청을 평가할수 있도록 의뢰기의 신원을 림시적으로 가정할수 할수 있다. 접근이 끝나면 봉사기는 의뢰기의 신원을 원래상태로 복귀한다.

접근통표

그림 15-11 ㄱ에서는 다음의 파라메터를 포함하는 접근통표의 일반구조를 보여 주고 있다.

- **보안 ID :** 망의 모든 기계들에서 유일하게 사용자를 식별한다. 이것은 일반적으로 사용자의 가입이름에 대응한다.
- **그룹 SID들 :** 사용자가 속하는 그룹들의 모임. 그룹은 단순히 접근조종을 위하여 그룹으로서 식별되는 사용자 ID들의 모임이다. 매개 그룹은 유일한 그룹 SID를 가진다. 객체에 대한 접근은 그룹 SID들, 개별적인 SID들 또는 그 결합에 기초하여 정의할수 있다.
- **특권들 :** 사용자가 호출할수 있는 보안에 민감한 체계봉사들의 목록. 실례로 생성통표를 들수 있다. 다른 실례로 모임여벌특권을 들수 있다. 이 특권을 가진 사용자들은 보통 읽을수 없는 파일들을 여벌로 작성하기 위한 여벌작성도구를 사용할수 있다. 대부분의 사용자들은 특권들을 전혀 가질수 없다.
- **기정소유자 :** 프로쎄스가 다른 객체를 생성하면 이 마당은 누가 새로운 객체의 소유자인가를 규정한다. 일반적으로 새로운 프로쎄스의 소유자는 부모프로쎄스의 소유자와 같다. 그러나 사용자는 프로쎄스가 만들어 낸 임의의 프로쎄스들의 기정소유자가 이 사용자가 속하는 그룹 SID이라는것을 규정할수 있다.
- **기정 ACL :** 이것은 사용자가 생성하는 객체들에 적용된 보호들의 초기목록이다. 사용자는 후에 자기가 소유하고 있거나 자기 그룹들중 하나의 그룹이 소유하고 있는 임의의 객체에 대한 ACL을 교체할수 있다.

보안 ID (SID)
그룹 SID 들
특권들
기정소유자
기정 ACL

ㄱ)

기발들
소유자
체계접근 조종목록(SACL)
선택접근조종목록 (DACL)

ㄴ)

ACI 머리부
ACE 머리부
접근마스크
SID
ACE 머리부
접근마스크
SID
⋮

ㄷ)

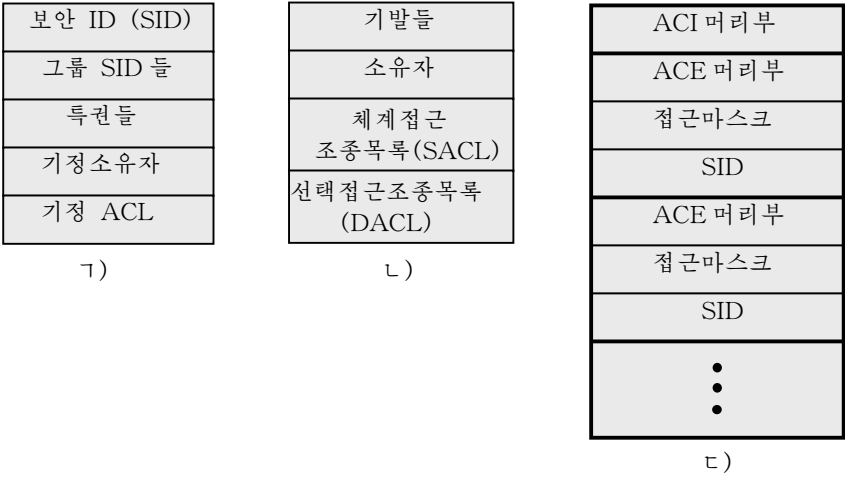



그림 15-11. Windows 2000 의 보안구조
ㄱ-접근통표, ㄴ-보안서술자, ㄷ-접근조종목록

보안서술자

그림 15-11 ㄴ에서는 다음의 파라메터를 포함하는 보안서술자의 일반구조를 보여 주고 있다.

- **기발들 :** 보안서술자의 종류 또는 내용들을 정의한다. 기발들은 SACL과 DACL이 존재하는가, 그것들이 기정기구에 의하여 객체에 배치되였는가 그리고 서술자에 있는 지시자들이 절대 또는 상대주소지정을 사용하는가를 지적한다. RPC로 전송된 정보와 같이 망을 통하여 전송되는 객체들에는 관련서술자들이 필요하다.
- **소유자 :** 객체소유자는 일반적으로 보안서술자에 대하여 임의의 작용을 수행할수 있다. 소유자는 개별적인 SID 또는 그룹 SID일수 있다. 소유자는 DACL의 내용을 변화시킬 권한을 가진다.
- **체계접근조종목록(SACL) :** 객체에 대한 어떤 종류의 조작들이 검열통보문들을 발생시키는가를 규정한다. 응용프로그람은 임의의 객체의 SACL을 읽거나 쓰기 위하여 대응하는 특권을 자기의 접근통표에 가지고 있어야 한다. 이것은 권한받지 못한 응용프로그람들이 SACL들을 읽지 못하게 하거나(이것으로 하여 검열통보문들의 발생을 피하기 위하여 무엇을 하지 말아야 할것인가를 알수 있다.) 그것들에 쓰지 못하게(위법적인 조작이 눈에 띄우지 않도록 많은 검열통보문들을 발생시키기 위해) 하기 위해서이다.
- **선택접근조종목록(DACL) :** 어떤 사용자들과 그룹들이 어떤 조작들을 위하여 객체에 접근할수 있는가를 결정한다. 그것은 접근조종입구들(ACE)의 목록으로 구성된다.

객체가 생성될 때 객체를 생성하는 프로쎄스는 자기의 접근통표에 자기소유의 SID 또는 임의의 그룹 SID를 소유자로 지정할수 있다. 생성하는 프로쎄스는 현재 접근통표에 없는 소유자를 지정할수 없다. 후에 객체의 소유자를 변화시킬 권한을 부여받는 임의의 프로쎄스는 그렇게 할수 있지만 반면에 그와 류사한 제한을 받는다. 제한하는 리유는 사용자가 어떤 허용되지 않은 작용을 한후에 자기 흔적을 없애 버리지 못하게 하기 위해서이다.

접근조종목록들은 W2K접근조종목록기능에서 기본이기때문에 그것들의 구조를 더 상세히 보자(그림 15-11 ㄷ). 매개 목록은 종합적인 머리부와 일정하지 않은 수의 접근조종입구점들로 구성된다. 매개 입구점은 개별 또는 그룹 SID를 규정하며 이 SID에 허가되는 권한들을 정의하는 접근마스크를 규정한다. 프로쎄스가 객체에 접근하려고 할 때 W2K집행부에 있는 객체관리자는 접근통표로부터 SID와 그룹 SID들을 읽고 객체의 DACL을 주사한다. 일치를 발견하면 즉 ACE에서 접근통표의 SID들중 한개와 일치하는 SID를 발견한다면 프로쎄스는 ACE의 접근마스크가 규정한 접근권한들을 가진다.

그림 15-12에서는 접근마스크의 내용들을 보여 주고 있다. 가장 낮은 쪽 16bit는 특수한 형태의 객체에 적용하는 접근권한들을 규정한다. 실례로 파일객체에서 비트 0은 File_Read_Data접근이며 사건객체에서 비트 0은 EventeQuery_Status접근이다.

마스크의 가장 높은 쪽 16bit는 모든 종류의 객체들에 적용하는 비트들을 포함한다. 이것들중 5개 비트를 표준접근형태라고 한다.

- **동기 :** 이 객체와 관련된 어떤 사건과 집행을 동기시키도록 허가한다. 특히 객체는 기다림기능에 사용될수 있다.

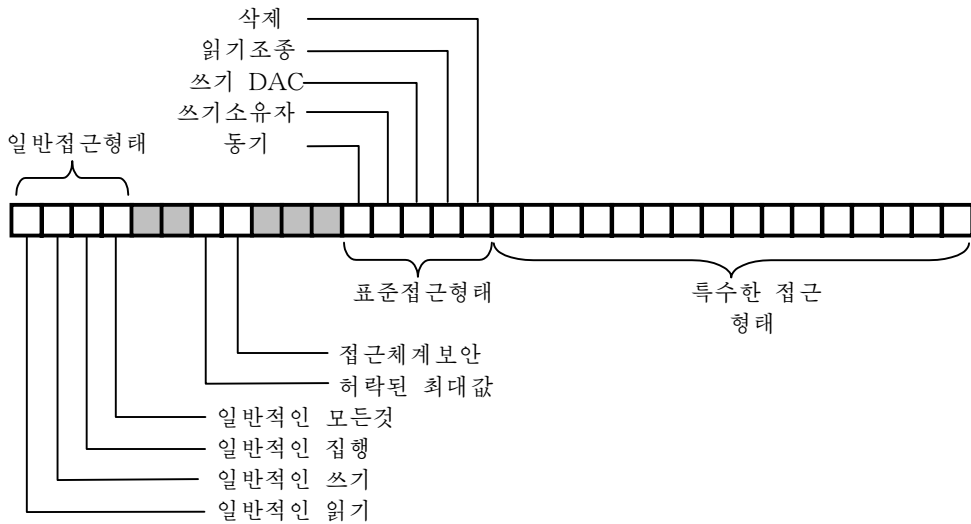



그림 15-12. 접근마스크

- **쓰기소유자 :** 프로그람이 객체의 소유자를 변경시킬수 있게 한다. 이것은 객체의 소유자가 항상 객체에 대한 보호를 변경시킬수 있기때문에 쓸모가 있다(소유자는 쓰기 DAC접근을 거절하지 않을수 있다.).
- **쓰기 DAC :** 응용프로그람이 DACL 즉 객체에 대한 보호를 변경시킬수 있게 한다.
- **읽기조종 :** 응용이 객체의 보안서술자의 소유자 및 DACL마당들을 문의하도록 한다.
- **삭제 :** 응용프로그람이 객체를 삭제하도록 한다.

접근마스크의 높은 자리 비트중 절반은 역시 4개의 일반접근형태들을 포함한다. 이 비트들은 여러개의 서로 다른 객체형태들에서 특정한 접근형태들을 설정하기 위한 편리한 방법을 준다. 실례로 응용프로그람이 여러가지 형태의 객체를 생성하려고 하며 사용자들이 객체들에 대한 읽기접근을 확실히 가진다고 가정하자. 일반접근비트들이 없이 매개 형태의 모든 객체를 보호하기 위하여 응용프로그람은 매개 형태의 객체에 서로 다른 ACE를 구축하여야 하며 매개 객체를 생성할 때 정확한 ACE를 주의하여 넘겨 주어야 한다. 일반개념을 표시하는 단일 ACE를 생성하고 이 ACE를 생성되는 매개 객체에 단순히 적용하며 옳은 사건이 발생하도록 하는것이 더 편리하다. 이 목적은 다음과 같은 일반접근비트들에 의해 실현된다. 즉

- 일반적인 모든것 : 모든 접근을 허락한다.
- 일반적인 집행 : 집행가능하다면 집행을 허락한다.
- 일반적인 쓰기 : 쓰기접근을 허락한다.
- 일반적인 읽기 : 읽기접근만을 허락한다.

일반비트들은 또한 표준접근형태들에 영향을 준다. 실례로 파일객체에 대하여 일반적인 읽기비트는 표준비트들인 읽기조종 및 동기비트에로 그리고 객체전용의 비트들인

File_Read_Data와 File_Read_Attribute, File_Read_EA에로 사영한다. SID에 일반적인 읽기를 허용하는 파일객체에 ACE를 할당하는것은 이 다섯개 접근권들을 접근마스크에서 개별적으로 규정한것처럼 허가한다.

접근마스크에서 나머지 두개 비트는 특별한 의미를 가진다. 접근체계보안비트는 이 객체에 대한 검열 및 경보조종을 변경하도록 허용한다. 그러나 이 비트는 어떤 SID의 ACE에 설정되여야 할뿐아니라 그 SID를 가지고 있는 프로쎄스에 대한 접근통표는 허가된 대응특권을 가져야 한다.

마지막으로 허락된 최대값비트는 실제상 접근비트는 아니지만 SID에 대한 DACL을 주사하기 위하여 W2K알고리듬을 변경시키는 비트이다. 일반적으로 W2K는 요청프로쎄스가 요청한 접근을 특별히 허가(비트설정) 또는 거절(비트비설정)하는 ACE를 찾을 때까지 또는 DACL의 끝에 이를 때까지 DACL을 주사한다. 후자의 경우에 접근은 거절된다. 허락된 최대값비트는 객체의 소유자가 주어 진 사용자에게 허가될 최대값인 접근권한들의 모임을 정의하도록 한다. 이것을 념두에 두고 응용프로그람은 대화기간에 객체를 실행하기 위하여 문의하게 될 모든 조작들을 알지 못한다고 가정하자. 접근을 요청하기 위한 방법에는 세가지가 있다. 즉

1. 모든 가능한 접근들에 대하여 객체를 열어 보시오. 이 방법의 결함은 응용프로그람이 대화에서 실제로 요구되는 모든 접근권들을 가질수 있다고 해도 접근이 거절될수 있다는것이다.
2. 특정한 접근이 요청될 때에만 객체를 연다. 그리고 서로 다른 모든 형태의 요청에 대하여 객체에로의 새로운 조종을 연다. 이것은 일반적으로 불필요하게 접근을 거절하지 않으며 또한 필요이상의 접근을 허용하지 않으므로 우월한 방법이다. 그러나 추가적인 내부조작이 필요하다.
3. 객체가 이 SID를 허락할만큼의 많은 접근에 대하여 객체를 열어 본다. 이 방법의 우점은 사용자의 접근을 인위적으로 거절하지 않지만 응용프로그람은 자기가 요구하는것보다 더 많이 접근권을 가진다는것이다. 후자의 상태는 응용프로그람에서 착오를 막을수 있다.

W2K보안의 중요한 특징은 사용자정의객체들에 W2K보안틀을 사용할수 있다는것이다. 실례로 자료기지봉사기는 자기의 보안서술자들을 생성하고 그것들을 자료기지의 부분들에 련결한다. 표준적인 읽기/쓰기접근수속들외에 봉사기는 결과모임의 흘리기 또는 결합의 수행과 같은 자료기지특유의 조작들을 보증할수 있다. 특수권한들의 의미를 정의하고 접근검사들을 수행하는것은 봉사기가 한다. 그러나 검사들에서는 표준문맥에서 발생하여 체계규모의 사용자/그룹회계들과 검열일지들을 사용한다. 확장된 보안모형은 외래파일체계의 실현들에 유익하게 쓸수 있다.

요약, 기본용어 및 복습문제

기관이 직면한 여러가지 보안위협들을 조사하면 보안에 대한 요구들을 가장 잘 평가할수 있다. 봉사의 중단은 사용성에 대한 위협으로 된다. 정보의 도청은 비밀엄수에 대한 위협으로 된다. 마지막으로 합법적인 정보의 변경과 정보의 승인되지 않은 날조는 완정성에 대한 위협으로 된다.

콤퓨터보안의 한가지 기본령역은 기억기보호이다. 이것은 여러개의 프로쎄스들이 동시에 동작하는 임의의 체계에서 기본이다. 가상기억기기구는 표준적으로 기억기보호를 위한 적당한 기구들로 장비된다.

다른 중요한 보안수법은 접근조종이다. 접근조종의 목적은 오직 권한 받은 사용자들만 특정한 체계와 그의 개별적자원들에 접근할수 있으며 자료의 특정부분들에로의 접근과 변경은 권한 받은 개체들과 프로그람들에 한정된다는것을 보증하는것이다. 엄격히 말하면 접근조종은 망보안문제라기보다 콤퓨터보안이다. 즉 대부분의 경우에 접근조종기법들은 단일콤퓨터에 실현되여 그 콤퓨터에 대한 접근을 조종한다. 그러나 콤퓨터에 대한 많은 접근은 망 또는 통신기구에 의한것이기때문에 접근조종기법은 분산망환경에서 효률적으로 운영할수 있도록 설계되여야 한다.

점점 더 걱정되는 형태의 위협은 비루스들과 그와 류사한 쏘프트웨어기법들에 의하여 제기되는 위협이다. 이 위협들은 체계쏘프트웨어의 약점을 사용하여 정보에 대한 승인되지 않은 접근권을 얻든가 체계봉사률을 떨어 뜨린다.

군사 및 상업환경들에서 점차 응용되고 있는 기술은 믿음직한 체계이다. 믿음직한 체계는 누가 어디에 접근하도록 권한을 받았는가에 따라 자료에 대한 접근을 조절하는 방법들을 제공한다. 기본문제는 체계가 주어 진 보안방책을 수행할것이라는것을 사용자가 완전히 믿을수 있도록 체계를 설계하고 실현하는것이다.

기본용어

접근조종	봉사거절	통과암호
적극적위협	암호화	공개열쇠암호화
고등암호화규격(AES)	전자우편비루스	재연
인증성	완정성	RSA(Rivest-Shamir-Adleman)
사용성	침입자	함정문
기밀성	침입검출	3중DEA
전통적인 암호화	론리폭탄	트로이목마
자료암호알고리듬 (DEA)	마크로비루스	믿음직한 체계
자료암호화규격 (DES)	비법적인쏘프트웨어 (멀웨어)	비루스
	피동위협	벌레
		좀비

복습문제

1. 콤퓨터보안에서 강조되는 기본요구들은 무엇인가?
2. 피동보안위협과 능동보안위협의 차이는 무엇인가?
3. 피동 및 능동보안위협들의 형태를 제시하고 간단히 정의하시오.
4. 가장 공통적인 사용자접근조종수법들에는 어떤 요소들이 필요한가?
5. 접근조종에서 주동체와 객체의 차이는 무엇인가?
6. 그림 15-5에서 양념의 목적을 설명하시오.
7. 통계적이상침입검출과 규칙에 기초한 침입검출의 차이를 설명하시오.
8. 1999년과 2000년의 전자우편부속프로그람과 전자우편 VBS비법쏘프트웨어제품들(실례로 Melissa, loveletter)을 매체에서 전자우편비루스라고 한다. 전자우편벌레라고 하는것이 더 정확하지 않는가?
9. 암호화는 비루스들의 설계에서 어떤 역할을 노는가?
10. 전통적인 암호화기구를 공격하기 위한 두가지 일반적인 방법은 무엇인가?
11. DES와 3중 DES란 무엇인가?

12. AES가 3중 DES보다 어떻게 되여 개선된것이라고 볼수 있는가?
13. AES후보자들을 평가하는데서 어떤 평가기준을 사용하는가?
14. 전통적인 암호화와 공개열쇠암호화의 차이는 무엇인가?
15. 용어 공개열쇠, 비공개열쇠, 비밀열쇠의 차이는 무엇인가?

참 고 문 헌

[STAL98]은 이 장의 주제에 대하여 더 상세하게 포괄하고 있다. 암호알고리듬들의 포괄범위에 있어서 [SCHN96]은 기본참고서로 된다. 이것은 지난 15년간에 공개된 실제상 모든 암호알고리듬과 규약을 서술하고 있다. 조작체계문제들에 대한 론의는 [BOLL99], [PILE 97], [SINH97], [SING94]에서 찾아 볼수 있다. [HOFF90]과 [DENN90]에서는 침입자와 비루스들과 관련된 많은 주요론문들을 전재하고 있다. [NACH97]은 항비루스기술의 최근동향에 대하여 설명하고 있다. [SHEL97]과 [SUTT97]에서는 Windows NT보안에 대하여 상세하게 설명하고 있다. 관리와 사용에 초점을 두면서 보안과 관련된 NT내부의 몇가지 특징들을 취급하고 있다.

BOLL99 Gollmann, D. *Computer Security.* New York: Wiley, 1999.

DENN90 Denning, P. *Computers Under Attack: Intruders, Worms, and Viruses.* Reading, MA:Addison-Wesley, 1990.

HOFF90 Hoffman. L., editor. Rogue Programs: Viruses, Worms, and Trojan Horses. New York: Van Nostrand Reinhold, 1990.

NACH97 Nachenberg, C. "Computer Virus-Antivirus Coevolution." *Communications of the A CM.* January 1997.

PFLE97 Pfleeger, C. *Security in Computing.* Upper Saddle River, NJ: Prentice Hall PTR, 1997.

SCHN96 Schneier, B. *Applied Cryptography.* New York: Wiley 1996. SCHN99 Schneier, B. "The Trojan Horse

SHEL97 Sheldon, T. *Window. NT Security Handbook.* New York: Osborne Mcgraw-Hill, 1997.

SING94 Singhat, M., and Shivaratri, N. *Advanced Concepts in Operating Systems.* New York:McGraw-Hill, 1994.

SINH97 Sinha, P. *Distributed Operating Systems.* Piscataway, NJ: IEEE Press, 1997. SMIT82 Smith, A. "Cache

STAL98 Stallings. W. *Cryptography and Network Security: Principles and Practice, 2nd ed.* Upper Saddle River. NJ: Prentice Hall, 1995.

SUTT97 Sutton, S. *Windows NT Security Guide.* Reading MA: Addison-Wesley. 1997.

련 습 문 제

1. 26개 자모문자중 4개 문자결합으로 통과암호를 선택한다고 가정하자. 적수가 초당 한번씩 통과암호를 시도할수 있다고 가정하자.

   ㄱ) 매개 시도가 끝날 때까지는 적수에 대한 반결합이 전혀 없다고 가정하면 정확한 통과암호를 발견하는 예상시간은 얼마인가?

ㄴ) 매개 부정확한 문자가 입구되는것과 같은 오유를 알리는 적수에로의 반결합을 가정한다면 정확한 통과암호를 발견하는 예상시간은 얼마인가?

2. 길이가 k인 원천요소가 어떤 통일적인 방식으로 길이가 p인 목적요소에 사영된다고 가정하자. 매개 수자가 r개 값들중 한개를 취한다면 원천요소들의 수는 r^k이며 목적요소들의 수는 그보다 작은수 i이다. 특정한 원천요소 x_i는 특정한 목적요소 y_i 에 사영된다.

ㄱ) 적수가 한번의 시도로 정확한 원천요소를 선택할수 있는 확률은 얼마인가?
ㄴ) 적수가 같은 목적요소 y_I로 되는 서로 다른 원천요소 $x_k(x_i \neq x_k)$를 산생할수 있는 확률은 얼마인가?
ㄷ) 적수가 한번의 시도로 정확한 목적요소를 산생할수 있는 확률은 얼마인가?

3. 어음통과암호발생기가 매개 6개 문자통과암호에서 임의로 두개 토막을 취한다. 매개 토막의 형식은 CVC(자음, 모음, 자음)이다. 여기서 V=〈a, e, i, o, u〉이고 C=$\overline{V}$ 이다.

ㄱ) 전체 통과암호수는 얼마인가?
ㄴ) 적수가 통과암호를 정확히 추측하는 확률은 얼마인가?

4. 통과암호로 95개의 인쇄가능한 ASCII문자들을 사용하도록 제한하고 모든 통과암호들의 길이가 10문자라고 가정하자. 초당 640만개의 암호화속도를 가진 통과암호해독자를 가정하자. UNIX체계에서 모든 가능한 통과암호들을 완전히 검사하는데 얼마만한 시간이 걸리겠는가?

5. UNIX통과암호체계에 대한 공개된 위협들로 하여 SunOS-4.0문서화에서는 통과암호파일이 제거되고 /etc/publickey라는 공개적으로 읽기가능한 파일로 교체된다고 권고한다. 사용자 A의 파일에로의 입구점은 사용자식별자 ID_A, 사용자공개열쇠 KU_a, 대응하는 비공개열쇠 KR_a로 구성된다. 이 비공개열쇠는 사용자가입통과번호 P_a로부터 넘겨 받은 열쇠와 함께 DES를 사용하여 암호화된다. A가 가입할 때 체계는 $EP_a[KR_a]$을 해독하여 KR_a를 얻는다.

ㄱ) 이때 체계는 P_a가 정확히 전달되였는가를 확인한다. 어떻게 확인하는가?
ㄴ) 적수가 이 체계를 어떻게 공격할수 있는가?

6. UNIX통과암호들에 사용된 암호화기구는 한 방향이다. 거꾸로는 불가능하다. 그러므로 이것은 사실상 통과암호의 암호화라고 말하기보다는 하쉬코드라고 말하는것이 정확하지 않겠는가?

7. UNIX통과암호기구에 양념을 포함시킨것은 추측난도를 4096배로 되게 한다고 서술되였다. 그러나 양념은 대응하는 암호문통과암호와 마찬가지로 같은 입구점에 평문으로 보관된다. 그러므로 이 두개 문자들은 공격자에게 알려 지거나 추측되지 말아야 한다. 양념이 왜 보안을 증가시킨다고 말하는가?

8. 앞의 문제에 정확히 대답하고 양념에 대하여 충분히 리해하였다고 가정한다면 또다른 질문이 제기된다. 양념의 크기를 례컨데 24 또는 48bit로 동적으로 증가시킨다면 모든 통과암호해독자들의 공격을 완전히 좌절시킬수 있지 않겠는가?

9. 여러준위보안체계에서 《읽기금지》규칙의 필요성은 매우 명백하다. 《쓰기금지》규칙의 중요성은 무엇인가?

10. 그림 15-10에서 트로이목마의 복사 및 후관측사슬의 한개 련결이 파괴된다. 알리스에 의한 두가지 서로 다른 가능한 공격방향이 존재한다. 즉 알리스는 가입하여 문자렬을 직접 읽으려고 하며 뒤주머니파일에 중요한 보안준위를 할당한다. 이때 참조감시기는 이 공격을 막아 낼수 있는가?

11. 누군가가 당신들중 두명이 같은 비밀열쇠를 소유하고 있다는것을 확인하기 위한 다음의 방법을 제안한다고 가정하자. 당신이 먼저 열쇠길이의 임의의 비트문자렬을 생성하고 그것을 열쇠와 XOR하고 결과를 통로로 송신한다. 당신들의 상대자가 들어 오는 블로크를 열쇠(당신의 열쇠와 류사한)와 XOR하고 그것을 되송신한다. 당신이 그것을 검사하고 수신한것이 자기의 원래의 임의의 비트문자렬이면 당신들중 누구도 열쇠를 전송한적이 없다고 해도 당신들은 상대자와 같은 비밀열쇠를 가진다는것을 확인하였다. 이 기구에 결함이 있는가?

부록 15-ㄱ. 암호화

실제상 모든 자동화된 망과 콤퓨터보안응용들의 기초로 되는 주요수법은 암호들이다. 두가지 기본방법 즉 대칭암호로서 알려 진 전통적인 암호화와 비동기암호화로 알려 진 공개열쇠암호화가 사용되고 있다. 부록에서는 몇가지 중요한 암호화알고리듬들에 대하여 간단히 설명하는것과 함께 두가지 암호화류형을 개괄한다.

전통적인 암호화

동기암호화 또는 단일열쇠암호화라고도 하는 전통적인 암호화는 1970년대말에 공개열쇠암호가 출현하기전에 사용하여 온 유일한 암호화방법이였다. 전통적인 암호화는 로마의 Julius Caesar부터 2차세계대전시기 도이췰란드의 U-잠수함부대와 오늘날의 외교, 군사, 상업부분의 사용자들에 이르기까지 헤아릴수 없이 많은 개인들과 그룹들에 의한 비밀통신에 사용되였다. 그것은 두가지 류형의 암호화가운데서 훨씬 더 광범하게 사용되고 있다.

전통적인 암호화기구는 다섯가지 요소를 가진다(그림 15-13).

- **평문 :** 이것은 알고리듬의 입력으로 공급되는 원본통보문 또는 자료이다.
- **암호알고리듬 :** 암호알고리듬은 평문에 대한 여러가지 치환과 변환을 수행한다.
- **비밀열쇠 :** 비밀열쇠도 역시 암호알고리듬의 입력이다. 알고리듬이 수행하는 정확한 치환과 변환들은 열쇠에 관계된다.
- **암호문 :** 이것은 출력으로 산생된 부호화된 통보문이다. 그것은 평문과 비밀열쇠에 관계된다. 주어 진 통보문에 대한 두개의 서로 다른 열쇠들은 두개의 서로 다른 암호문을 산생한다.
- **암호해제알고리듬 :** 이것은 본질상 반대로 실행되는 암호알고리듬이다. 그것은 암호문과 비밀열쇠를 가지고 원래의 평문을 산생한다. 전통적인 암호화를 안전하게 사용하기 위한 두가지 필요조건이 있다. 즉

1. 우리에게는 강력한 암호알고리듬이 필요하다. 우리는 최소한 알고리듬을 알고 있고 한개이상의 암호문에 접근하는 적수가 암호문을 해독하거나 열쇠를 해석할수 없는 알고리듬이 요구된다. 이 요구는 더욱 강경하게 제기된다. 즉 적수가 비록 여러개의 암호문과 함께 매개 암호문을 생성한 평문을 입수한다고 해도 암호문을 해독하거나 열쇠를 발견할수 없을것을 요구한다.

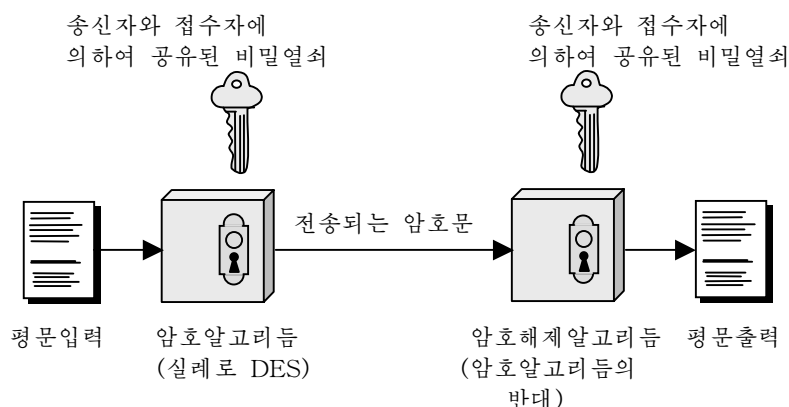



그림 15-13. 전통적인 암호화기구

2. 송신자와 수신자는 안전한 방식으로 비밀열쇠의 사본들을 획득하며 열쇠를 안전하게 유지하여야 한다. 누군가가 열쇠를 발견하고 알고리듬을 알수 있다면 이 열쇠를 사용하는 모든 통신은 쉽게 해독된다.

전통적인 암호화기구를 공격하기 위한 두가지 일반적인 방법이 있다. 첫번째 공격방법은 **암호해석학**으로 알려 져 있다. 암호해석공격들은 알고리듬의 특징과 함께 평문의 일반특성들에 대한 어떤 지식이나 지어 어떤 표본적인 평문-암호문쌍들에 관계된다. 이런 형태의 공격은 알고리듬의 특성들을 사용하여 특정한 평문을 추론하거나 사용되고 있는 열쇠를 추론해 본다. 열쇠에 대한 추론공격에서 성공하면 결과는 파국적이다. 그 열쇠로 암호화되는 미래와 과거의 모든 통보문들은 위태롭게 된다.

열쇠전수공격법으로 알려 진 두번째 방법은 평문으로 리해하기 쉽게 변환될 때까지 부분적인 암호문에 대하여 모든 가능한 열쇠들을 시도하는것이다. 평균하여 모든 가능한 열쇠들의 절반을 시도하여 성공하여야 한다. 표 15-5에서는 여러가지 열쇠크기에 대하여 얼마만한 시간이 요구되는가를 보여 준다(표는 한개의 암호화를 수행하는데 1㎲ 걸린다고 가정하고 오늘날의 콤퓨터에 알맞는 자리수의 매개 열쇠크기에 대한 결과를 보여 준다.). 표는 많은 극소형처리기들로 병렬구성된 체계를 사용하는 경우에 보다 높은 자리의 처리속도를 달성할수 있다. 표의 마지막렬은 마이크로초당 백만개 열쇠를 처리할수 있는 체계에서의 결과를 보여 준다. 한가지 알수 있는것은 이 성능준위에서 56bit열쇠가 계산상 더는 안전하다고 볼수 없다는것이다.

표 15-5. 열쇠전수탐색에 필요한 평균시간

열쇠길이 (bit)	가능한 열쇠수	1암호/㎲속도에서 필요한 시간	10^6암호/㎲속도에서 필요한 시간
32	2^{32}=4.3×10^9	2^{31} μs=35.8분	2.15ms
56	2^{56}=4.3×10^{16}	2^{55} μs=1142년	10시간
128	2^{128}=4.3×10^{38}	2^{127} μs=5.4×10^{24}년	5.4×10^{18}년
168	2^{168}=3.7×10^{50}	2^{167} μs=5.9×10^{36}년	5.9×10^{30}년

자료암호화표준(DES)

가장 널리 사용되는 암호화기구는 국가규격표준국 지금은 미국규격 및 기술협회(NIST)에서 1977년에 미련방정보처리규격46(FIPS PUB 46)으로 채택된 자료암호화표준(DES)에서 정의된다. 1994년에 NIST는 앞으로 5년간 FIPS PUB 46-2에서 련방적으로 사용하기 위하여 DES를 개정하였다. NIST는 극비정보의 보호를 제외한 응용프로그람들에서 DES를 사용할것을 권고하였다. 알고리듬자체는 자료암호알고리듬(DEA)이라고 한다.

모든 암호화기구의 경우와 마찬가지로 DES암호화기능은 두가지 즉 암호화되는 평문과 열쇠를 요구한다. DES의 경우에 평문은 64bit이고 열쇠는 56bit이여야 한다. 보다 긴 평문블로크들은 64bit의 블로크들로 나누어 암호화된다.

본질에 있어서 DES에서는 매 반복의 결과로 64bit의 중간값을 발생시키면서 16회의 반복에 걸쳐 매개 64bit입력을 넘겨 주는 방법으로 평문을 처리한다. 매개 반복은 본질상 비트들의 치환과 한비트패턴을 다른 비트패턴으로 바꾸는 동작을 동반하는 똑같이 복잡한 함수이다. 매개 단계의 입력은 이전 단계의 출력과 함께 보조열쇠라고 하는 열쇠비트들의 치환값으로 구성된다.

DES의 암호해제과정은 암호화와 본질상 동일하다. 즉 암호문을 DES알고리듬의 입력으로 사용한다. 그러나 매개 반복에서 발생한 보조열쇠들을 반대순서로 사용한다(실례로 첫번째 반복에는 열여섯번째 보조열쇠를, 두번째 반복에는 열다섯번째 보조열쇠를 사용).

DES의 강도

1970년대말에 전문가들은 안전한 알고리듬으로서의 DES시대는 얼마 남지 않았으며 처리기속도의 증가와 하드웨어원가의 저하로 DES가 무력해 지는것은 시간문제라고 경고하였다. 1998년 7월 전자첨단재단(EFF)이 25만딸라이하로 제작한 전용목적의 《DES해독기》기계를 사용한 새로운 DES도전은 실패하였다고 방송하였을 때 DES에게는 결국 죽음이 선언되였다. EFF는 기계의 상세한 설명을 공개하고 다른 사람들이 자기의 해독기를 구축할수 있게 하였다[EFF98]. 물론 하드웨어가격은 속도가 증가함에 따라 계속 떨어 지고 DES를 가치 없는것으로 만들것이다.

다행히도 시장에서 입수할수 있는 여러개의 다른 방법들이 있다. 다음에는 가장 광범히 사용되는 방법들을 보게 된다.

3중 DEA

열쇠전수공격법에 대한 DES의 약점이 알려 졌기때문에 다른 방법을 찾는데로 관심이 돌려 졌다. 현재의 투자를 쏘프트웨어와 장비에 보존하는 한가지 방법은 DES와 다중열쇠들에 의한 다중암호화를 사용하는것이다. 3중DEA(TDEA)는 1985년에 재정부분의 응용프로그람들에 사용하기 위하여 ANSI규격 X9.17에서 처음으로 표준화되였다. TDEA는 1993년에 자료암호화표준의 일부로 FIPS PUB 46-3의 공개판에 통합되였다.

TDEA는 3개의 열쇠를 사용하여 DES알고리듬을 세번 집행한다. 조작은 암호화-암호해제-암호화(EDE)순서로 진행된다.[2] TEDA에서 세개의 서로 다른 열쇠들은 168bit의 효률적인 열쇠길이를 가진다. FIPS46-3은 또한 K1=K3인 두개의 열쇠를 사용할수 있다. 이때 열쇠의 길이는 112bit이다. FIPS46-3에는 TDEA에 대한 다음의 지침들이 포함되여 있다. 즉

---

2. 두번째 단계에서 암호해제를 사용하는것은 암호에 아무런 의의도 없다. 그것의 유일한 우점은 3중 DES의 사용자들이 이전 단일사용자들에 의하여 세개의 매 단계에서 같은 열쇠를 단순히 반복하여 암호화한 자료를 암호해제하도록 한다는것이다.

- TDEA는 FIPS에 통과된 전통적인 암호알고리듬이다.
- 56bit의 단일열쇠를 사용하는 원래의 DEA는 오직 종전의 체계들을 위한 규격하에서 허가되였다. 지금 나오고 있는 체계들은 TDEA를 지원한다.
- 종전의 DEA체계들을 가지고 있는 정부기관들에 TDEA에로 이행할것이 권고되고 있다.
- TDEA와 고급암호화표준(AES)은 FIPS통과알고리듬으로서 공존하는데 AES에로 점차 이행할것이 예견된다.

TDEA는 방대한 알고리듬이라고 보기 쉽다. 기초를 이루는 암호알고리듬이 DEA이기때문에 TDEA는 DEA에 요구되는 알고리듬에 기초한 암호해석학에 똑같이 대항할것을 요구할수 있다. 게다가 168bit의 열쇠길이인 경우 열쇠전수공격은 실제상 불가능하다. DES의 약점들을 용납할수 있는 앞으로의 몇년동안 그리고 AES의 본격적인 전개가 시작될 때까지는 TDEA의 사용이 계속 증가될것이라고 기대할수 있다.

고급암호화표준

TDEA는 앞으로 두가지 리유로 하여 몇년동안 널리 사용할것으로 예견된다. 첫째로, 열쇠길이가 168bit인 경우 DEA의 열쇠전수공격에 대한 약점을 극복할수 있다. 둘째로, TDEA에서 기초로 되는 암호알고리듬은 DEA에서와 같은 알고리듬이다. 이 알고리듬은 오랜 기간에 걸쳐 다른 모든 암호알고리듬보다 더 자세히 연구되여 왔으며 이 알고리듬에 기초한 암호해석공격이 열쇠전수공격보다 비효률적이라는것을 알수 있었다. 그러므로 TDEA는 암호해석에 대하여 강하게 저항한다고 크게 믿을수 있다. 보안만을 념두에 둔다면 TDEA는 앞으로 몇십년동안 표준화된 암호알고리듬으로 될수 있을것이다.

TDEA의 주요한 약점은 알고리듬이 쏘프트웨어에서 상대적으로 속도가 뜬것이다. 원래의 DEA는 1970년대 중엽에 하드웨어적으로 실현하려고 설계되였으며 효률적인 쏘프트웨어코드를 만들어 내지 못하였다. DEA만한 반복의 3배가 되는 TDEA는 그만큼 더 뜨다. 두번째 약점은 DEA와 TDEA가 다 64bit의 블로크를 사용한다는것이다. 효률 및 보안문제와 관련해서는 보다 큰 블로크가 필요하다.

이 약점들때문에 TDEA는 오랜 기간 사용할수 있는 적당한 후보자가 못된다. 후보자로서 NIST는 1997년에 새로운 고급암호화표준(AES)에 대한 제안을 발표하였다. AES는 TDEA와 같거나 더 좋은 보안강도를 가지며 효률도 매우 개선되였다. 이 일반적인 요구들에 추가하여 NIST는 AES의 블로크길이가 128bit인 대칭블로크암호이여야 하며 길이가 128, 192, 256bit인 열쇠를 지원하여야 한다고 규정하였다. 평가기준에는 보안, 계산효률, 기억기요구, 하드웨어 및 쏘프트웨어의 적합성, 유연성이 포함되였다.

첫번째 평가단계에서는 15개의 제안된 알고리듬들이 접수되였다. 두번째 단계에서는 5개 알고리듬들로 줄어 들었다. 이 책을 집필하는 현재 NIST에서는 평가를 끝내고 2001년 여름까지 최종규격을 공개하려고 하고 있다. 시장진출은 그후 몇년이 걸릴수 있다.

공개열쇠암호화

Diffie와 Hellman에 의하여 1976년에 처음으로 공개적으로 제안된 공개열쇠암호화는 사실상 수천년동안 암 호화에서 처음으로 되는 진짜 혁신적인 진보였다. 우선 한가지 실례를 든다면 공개열쇠알고리듬은 단순한 비트패턴들의 조작에 기초한것이 아니라 수학적함수들에 기초하고 있다. 보다 중요한것은 공개열쇠암호에서 오직 한개의 열쇠만을 사용하는 대칭형의 전통적인 암호화와는 달리 비대칭으로서 두개의 독립적인 열쇠들을 사용하는것이다. 두개의 열쇠를 사용하면 기밀성, 열쇠배포, 인증의 령역들에서 심중한 결과를 가져 온다.

먼저 공개열쇠암호화와 관련되는 여러가지 공통적인 잘못된 견해들에 대하여 언급한다. 첫번째 잘못된 견해는 공개열쇠암호화가 암호해석에 대하여 전통적인 암호화보다 더 안전하다는것이다. 사실상 모든 암호화방안의 보안은 열쇠의 길이와 암호를 해석하는데 필요한 계산작업에 관계된다. 암호해석에 대항하는 관점에서 볼 때 전통적인 암호화나 공개열쇠암호화중에서 어느것이 더 우월한가 하는데는 원리적으로 특별한것이 없다. 두번째 잘못된 견해는 공개열쇠암호화가 전통적인 암호화를 쓸모 없게 만드는 범용수법이라는것이다. 그러나 현대적인 공개열쇠암호화방안들의 내부계산처리때문에 전통적인 암호화가 포기될 가능성은 전혀 있을것 같지 않다. 결국 공개열쇠암호화를 적용하면 전통적인 암호화에서 열쇠배포쎈터들을 필요로 하는 번잡한 맞잡이처리에 비하여 열쇠배포가 매우 쉬워진다. 사실상 중앙대리점을 포함하는 어떤 형식의 규약이 필요한데 이때 동반되는 수속들은 전통적인 암호화에 필요한 수속들보다 더 효률적이지도 더 단순하지도 않다.

공개열쇠암호화기구는 5개의 요소를 가진다(그림 15-14). 즉

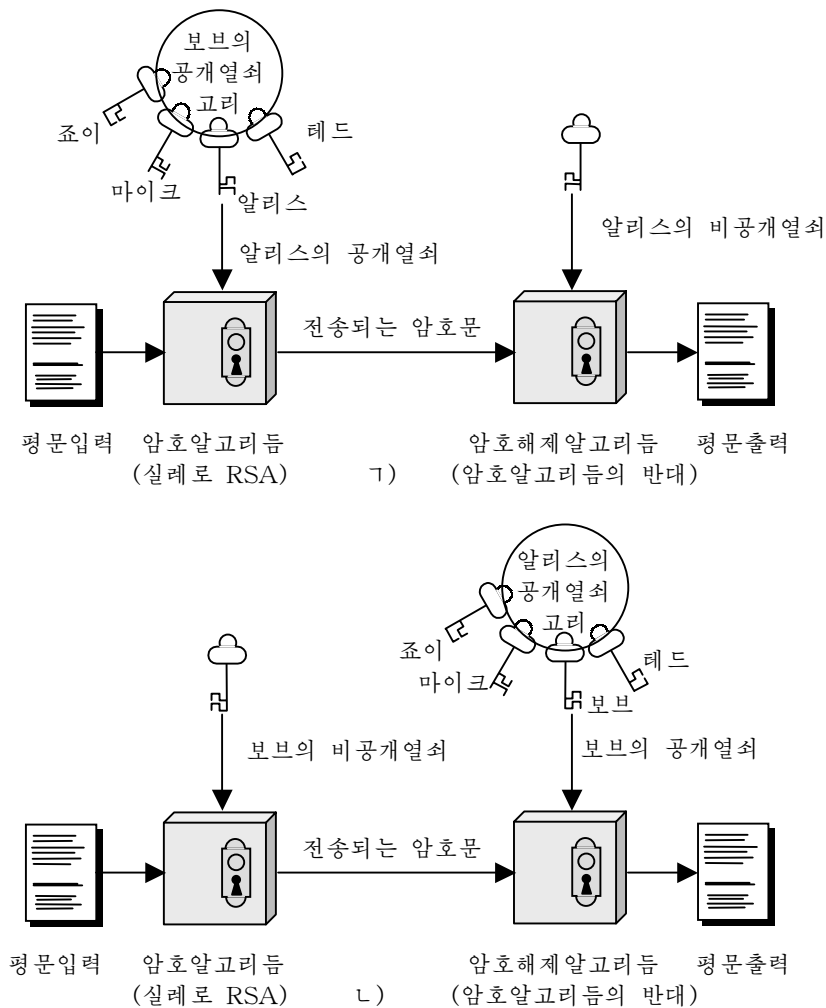



그림 15-14. 공개열쇠암호화
ㄱ-암호화, ㄴ-서명

- **평문 :** 알고리듬에 입력으로 공급되는 읽기가능한 통보문 또는 자료이다.
- **암호알고리듬 :** 암호알고리듬은 평문에 대한 여러가지 변환을 수행한다.
- **공개 및 비공개열쇠 :** 한개가 암호화에 사용되면 다른것은 암호해제에 사용되도록 선택된 열쇠쌍이다. 암호알고리듬이 수행하는 변환들의 정확성은 입력으로 주는 공개 또는 비공개열쇠에 관계된다.
- **암호문 :** 출력으로 산생된 변신된 통보문이다. 암호문은 평문과 열쇠에 관계된다. 주어 진 통보문에 대한 두개의 서로 다른 열쇠들은 두개의 서로 다른 암호문들을 산생한다.
- **암호해제알고리듬 :** 이 알고리듬은 암호문과 열쇠쌍의 다른 한쪽 열쇠를 입력하여 원래의 평문을 산생한다.

처리는 열쇠쌍이 사용되는 순서를 고려함이 없이 진행된다(정확한 평문을 출력으로 생성한다.). 이름들이 암시하는바와 같이 열쇠쌍에서 공개열쇠는 다른 사용자들이 사용할수 있도록 공개되며 비공개열쇠는 그 소유자만 안다.

이제 보브가 알리스에게 비밀통보문을 보내려고 한다고 하자. 그리고 보브는 알리스의 공개열쇠를 가지고 있으며 알리스는 공개열쇠와 쌍이 되는 비공개열쇠를 가지고 있다고 가정하자(그림 15-14 ㄱ). 보브는 알리스의 공개열쇠를 사용하여 통보문을 암호화하여 암호문을 생성한다. 그다음 암호문은 알리스에게 전송된다. 알리스는 암호문을 받으면 자기의 비공개열쇠를 사용하여 그것을 암호해제한다. 알리스만이 자기의 비공개열쇠 사본을 가지고 있기때문에 그밖의 누구도 그 통보문을 읽을수 없다.

공개열쇠암호화는 그림 15-14 ㄴ에서 설명된것과 같이 다른 방법으로 사용할수도 있다. 보브가 알리스에게 통보문을 보내려고 하며 통보문을 비밀로 하는것이 중요하지 않지만 보브는 알리스가 그 통보문이 정말로 자기한데서 오는것이다는것을 확신하기를 바란다고 가정하자. 이 경우에 보브는 자기소유의 비공개열쇠를 사용하여 통보문을 암호화한다. 알리스는 암호문을 수신하면 보브의 공개열쇠로 그것을 암호해제할수 있다는것을 알고 그 통보문이 보브가 틀림없이 암호화하였다는것을 확인한다. 그밖의 누구도 보브의 비공개열쇠를 가지고 있지 않으며 따라서 누구도 보브의 공개열쇠로 암호문을 생성할수 없다. 범용공개열쇠암호알고리듬에서는 한개의 열쇠로 암호화를 하고 다른 열쇠로 암호해제를 한다. 또한 이 알고리듬은 다음의 중요한 특징들을 가진다. 즉

- 암호알고리듬에 대한 지식만으로 주어 진 암호해제열쇠와 암호열쇠를 결정하는것은 계산상 불가능하다.
- 두개의 관련열쇠들중 어느 한쪽 열쇠는 암호화에, 다른 열쇠는 암호해제에 사용할수 있다.

기본단계는 다음과 같다. 즉

1. 매개 사용자는 통보문의 암호화와 암호해제에 사용되는 열쇠쌍을 발생한다.
2. 매개 사용자는 두개 열쇠중 한개를 공개등록기 또는 접근가능한 다른 파일에 보관한다. 이것이 바로 공개열쇠이다. 다른쪽 열쇠는 비밀로 보관한다. 그림 15-14 ㄱ에서 암시하는바와 같이 매개 사용자는 다른 사용자들한테서 받은 공개열쇠묶음을 가지고 있다.

3. 보브가 알리스에게 비밀통보문을 송신할 때 보브는 알리스의 공개열쇠를 사용하여 통보문을 암호화한다.
4. 알리스는 통보문을 수신하면 자기의 비공개열쇠를 사용하여 그것을 암호해제한다. 알리스만이 자기의 비공개열쇠를 알고 있기때문에 다른 접수자는 그 통보문을 전혀 암호해제할수 없다.

이 방법에서 모든 관계자들은 공개열쇠에 접근하고 비공개열쇠들은 매개 관계자에 의하여 국부적으로 발생되며 따라서 전혀 배포할 필요는 없다. 사용자가 자기의 비공개열쇠를 보호하는한 들어 오는 통신은 안전하다. 임의의 시각에 사용자는 비공개열쇠를 변경시킬수 있으며 낡은 공개열쇠를 교체하기 위하여 변경된 비공개열쇠에 대응하는 공개열쇠를 공개할수 있다.

전통적인 암호화에 사용되는 열쇠는 보통 비밀열쇠라고 한다. 공개열쇠암호화에 사용되는 두개의 열쇠를 각각 공개열쇠와 비공개열쇠라고 한다. 비공개열쇠는 변함없이 비밀로 보관되지만 전통적인 암호화와의 혼돈을 피하기 위하여 비밀열쇠가 아니라 비공개열쇠라고 한다.

리버스트 샤미어 아들레만(RSA)알고리듬

첫 공개열쇠방안들중의 하나가 1977년에 MIT의 론 리베스트, 애디 샤미어, 렌 아들레만에 의하여 개발되였다. RSA방안은 그이후 공개열쇠암호화에 가장 널리 도입되고 실현된 유일한 방법으로서 가장 널리 류행되고 있다. RSA는 평문과 암호문이 어떤 n에 대하여 0과 n-1사이에 있는 옹근수들로 된 암호이다. 암호화는 모듈산법을 동반한다. 알고리듬의 강도는 수자들을 소인수들로 분해하는데서 제기되는 곤난성에 관계된다.

부록 1. TCP/IP

이 부록에서는 먼저 계층화된 통신규약구성방식을 소개한다. 다음으로 계층화된 통신규약구성방식에서 가장 중요한 TCP/IP규약을 고찰한다. TCP/IP는 인터네트에 기초한 개념이며 모든 령역의 콤퓨터통신규격들을 개발하기 위한 기초구조이다. 실제상 모든 콤퓨터제작자들은 이러한 구성방식을 지원하고 있다.

1-1. 통신규약구성방식에 대한 요구

콤퓨터들과 말단들 그리고 다른 자료처리장치들이 자료를 교환할 때 동반되는 절차들은 아주 복잡할수 있다. 실례로 두 콤퓨터사이의 자료전송을 고찰하자. 두 콤퓨터사이에는 직접 또는 통신망을 통한 자료경로가 있어야 한다. 그러나 보다 많은 요구항목이 제기된다. 수행해야 할 대표적인 과제들로는 다음과 같은것들이 포함된다. 즉

1. 원천체계는 직접 자료통신경로를 활성화시키거나 희망하는 목적체계의 신원을 통신망에 통지하여야 한다.
2. 원천체계는 목적체계가 자료를 수신할 준비가 되여 있다는것을 확인하여야 한다.
3. 원천체계의 파일전송응용프로그람은 목적체계의 파일관리프로그람이 특정한 사용자를 위하여 파일을 접수하고 보관할 준비가 되여 있음을 확인하여야 한다.
4. 두 체계에서 사용되는 파일형식이 호환성이 없으면 한 체계 또는 다른 체계는 양식변환기능을 수행하여야 한다.

두 콤퓨터체계들사이에서 높은 급의 협동처리가 진행되여야 하는것은 명백하다. 이것을 단일모듈로 론리적으로 하지 않고 과제를 부분과제들로 나누고 매개 과제를 독립적으로 실현한다. 통신규약구성방식에서 모듈들은 가상탄창에 배렬된다. 탄창에 있는 매개 층은 다른 체계와 통신하는데 필요한 함수들의 관련부분모임을 수행한다. 매개 층은 자기보다 낮은 층에 의거하여 더 많은 기본지령함수들을 수행하며 이 함수들의 상세한 정보를 숨긴다. 매개 층은 자기보다 높은 층에 봉사한다. 층들은 리론상 한개 층에서의 변화가 다른 층들에서의 변화를 요구하지 않도록 설계되여야 한다.

물론 두 체계가 통신하므로 똑같은 계층화된 함수들의 모임이 두 체계에 있어야 한다. 두 체계가 대응하는 또는 같은 준위의 통신층들을 가져야 통신이 이루어 진다. 같은 준위의 층들은 통신규약이라는 규칙 또는 규약들의 모임에 따르는 양식화된 자료블로크로 통신한다. 통신규약의 기본특징은 다음과 같다. 즉

- **문장론** : 자료블로크의 양식과 관련된다.
- **의미론** : 동위어구와 오유조정을 위한 조종정보를 포함한다.
- **박자동기** : 속도의 조화와 순서화를 포함한다.

1-2. TCP/IP규약의 구성방식

TCP/IP규약구성방식은 고등연구계획청(DARPA)이 투자한 시험적인 파케트교환망 ARPANET에 대한 통신규약연구와 개발의 산물로서 일반적으로 TCP/IP규약이라고 한다. 이 규약은 인터네트방식국(IAB)이 인터네트규격들로 제출한 큰 규약집합으로 구성된다.

TCP/IP층

쉽게 말하면 통신은 세개의 중개자 즉 응용프로그람, 콤퓨터, 망을 동반한다고 말할수 있다. 응용프로그람의 실례로는 파일전송과 전자우편을 들수 있다. 여기서 론의하는 응용프로그람들은 두 콤퓨터체계들사이의 자료교환을 포함하는 분산응용프로그람들이다. 이 응용프로그람들과 그밖의 응용프로그람들은 흔히 여러개 응용프로그람들의 동시실행을 지원할수 있는 콤퓨터들에서 집행한다. 콤퓨터들은 망에 련결되며 교환될 자료는 한 콤퓨터에서 다른 콤퓨터로 망을 통하여 전송된다. 따라서 한 응용프로그람에서 다른 응용프로그람에로의 자료전송은 먼저 응용프로그람이 존재하는 콤퓨터에 자료를 주고 그다음 콤퓨터의 예정된 응용프로그람에로 자료를 주는 동작을 포함한다.

이 개념들을 념두에 두고 통신과제를 상대적으로 독립적인 다섯개의 층으로 구성한다.

- 물리층
- 망접근층
- 인터네트층
- 가입자 대 가입자층 또는 전송층
- 응용층

물리층은 자료전송장치(실례로 워크스테이션, 콤퓨터)와 전송매체 또는 망사이의 물리적대면부역할을 수행한다. 이 층은 전송매체의 특징, 신호들의 특성, 자료속도 그리고 그와 관련된 문제들을 규정한다.

망접근층은 말단체계(봉사기, 워크스테이션 등)와 그와 련결된 망사이의 자료교환과 관련된다. 송신콤퓨터가 목적콤퓨터의 주소를 망에 제공하여야 망이 적당한 목적지에로 자료의 경로를 지정할수 있다. 송신콤퓨터는 우선권과 같은 망이 제공하는 일정한 봉사들을 받으려고 할수 있다. 이 층에서 사용되는 특정한 쏘프트웨어는 사용되는 망의 형태에 관계된다. 즉 서로 다른 규격들이 회선교환, 파케트교환(실례로 X.25), LAN(실례로 에써네트) 및 다른것들을 위하여 개발되여 왔다. 따라서 망접근과 관계되는 기능들을 독립적인 층으로 분리하는것이 좋다. 이렇게 하면 망접근층우에 있는 통신쏘프트웨어의 나머지 부분은 사용되는 망의 특징들과 관련될 필요가 없다. 보다 높은 층의 같은 쏘프트웨어는 콤퓨터가 련결되는 특정한 망에 관계없이 적당히 기능을 수행할것이다.

망접근층은 같은 망에 련결된 두개의 말단체계의 망을 통하여 자료에 접근하고 자료의 경로를 지정한다. 두개의 장치가 서로 다른 망에 련결되는 경우에 자료가 다중상호련결망들을 관통하도록 하는 수속들이 요구된다. 이것이 **인터네트층**의 기능이다. 망간통신규약(IP)은 다중망에서의 경로결정기능을 제공하기 위하여 이 층에서 사용된다. 이 규약은 말단체계들에서뿐아니라 경로기에서도 실현된다. 경로기는 두개의 망을 련결하는 처리기이며 그의 1차기능은 원천말단체계에서 목적말단체계에로의 경로를 통하여 한 망에서 다른 망으로 자료를 중계하는것이다.

자료를 교환하는 응용프로그람들의 특징에는 관계없이 보통 자료는 믿음성있게 교환되여야 한다. 즉 모든 자료가 목적응용프로그람에 도착하며 그 자료는 송신된 순서와 같은 순서로 도착한다는것이 보증되여야 한다. 알수 있는바와 같이 믿음성을 제공하기 위한 기구들은 본질상 응용프로그람들의 특징과 무관계하다. 따라서 모든 응용프로그람들이 공유하는 공통층에 이 기구들을 집합시키는것이 합리적이다. 이 공통층을 바로 **가입자 대 가입자층 또는 전송층**이라고 한다. 전송조종통신규약(TCP)은 이 기능을 위하여 가장 공통적으로 사용되는 규약이다.

마지막으로 **응용층**은 여러가지 사용자응용프로그람들을 지원하는데 요구되는 론리들

을 포함한다. 파일전송과 같은 서로 다른 형태의 응용프로그람에서는 응용에 고유한 개별적인 모듈이 필요하다.

TCP와 IP의 조작

그림 부 1-1은 규약들이 통신을 위하여 어떻게 구성되는가를 보여 준다. 모든 통신기지들이 여러개의 망들로 구성될수 있다는것을 명백히 하기 위하여 구성하는 망들을 보통 부분망이라고 한다. 에써네트론리와 같은 어떤 종류의 망접근규약은 콤퓨터를 부분망에 련결하는데 사용된다. 이 규약으로서 가입자는 부분망을 통하여 다른 가입자에로 또는 다른 부분망에 있는 가입자인 경우에는 경로기에로 자료를 송신할수 있다. IP는 모든 말단체계들과 경로기들에서 실현된다. IP는 중계기로 동작하여 자료블로크를 하나 또는 그이상의 경로기를 통하여 한 가입자에서 다른 가입자에로 이동시킨다. TCP는 말단체계들에서만 실현된다. TCP는. 모든 자료가 적당한 체계에로 믿음직하게 전송된다는것을 보증하기 위하여 자료블로크들을 추적한다.

성과적인 통신을 위하여 전반체계에 있는 매개 실체는 유일한 주소를 가져야 한다. 실제로 두개의 주소지정준위가 필요하다. 부분망에 있는 매개 가입자는 유일한 전역인터네트주소를 가져야 한다. 이것은 자료가 적당한 가입자로 전송되도록 한다. 가입자의 매개 프로쎄스는 가입자내에서 유일한 주소를 가져야 한다. 이것은 가입자 대 가입자규약(TCP)이 자료를 적당한 프로쎄스에 전송하도록 한다. 후자의 주소를 포구라고 한다.

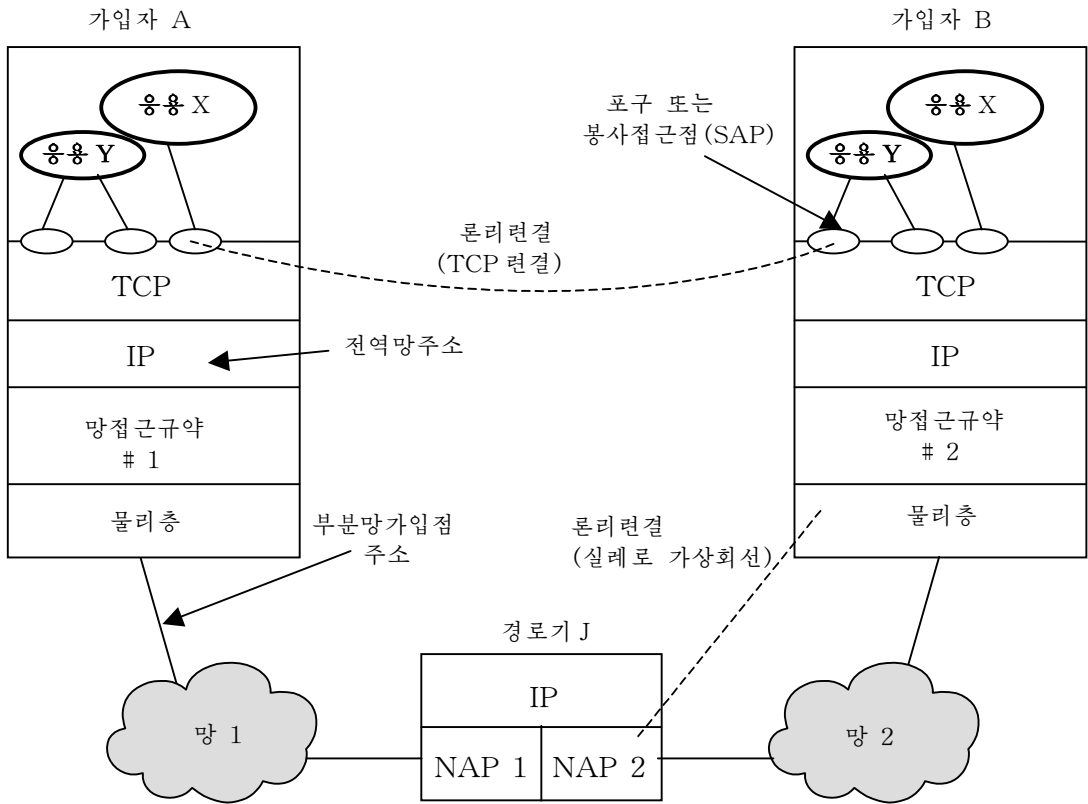



그림 부 1-1. TCP/IP의 개념

간단한 조작을 살펴 보자. 가입자 A의 포구 1과 관련된 프로쎄스가 통보문을 가입자 B의 포구 3과 관련된 프로쎄스에 송신하려고 한다고 가정하자. A의 프로쎄스는 통보문을 가입자 B의 포구 3에로 송신하기 위하여 명령들과 함께 TCP에 넘겨 준다. TCP는 통보문을 가입자 B로 송신하기 위하여 명령문과 함께 IP에 넘겨 준다. IP에 목적포구의 신원을 알려 줄 필요는 없다. IP가 알고 싶어 하는것은 가입자 B로 보내 지는 자료만이다. 다음으로 IP는 통보문을 경로기 J(B로 가는 경로의 첫 홉)에로 송신하기 위하여 명령들과 함께 망접근층(실례로 에써네트론리)에 넘겨 준다.

이 조작을 조종하기 위하여 사용자자료는 물론 조종정보는 그림 부1-2에 제시된것과 같이 전송되여야 한다. 송신프로쎄스는 자료블로크를 발생하여 이것을 TCP로 넘긴다. TCP는 이 블로크를 더 작은 부분들로 쪼개여 조작하기 쉽게 한다. 이 매개 부분들에 대하여 TCP는 TCP머리부라는 조종정보를 달아 TCP토막을 만든다. 조종정보는 가입자 B의 동일한 TCP규약실체가 사용한다. 머리부의 항목에 대한 실례로서 다음과 같은것을 볼수 있다.

- **목적포구** : B의 TCP실체는 토막을 수신할 때 자료가 누구한테서 전송되는가를 알아야 한다.
- **순차번호** : TCP는 특정한 목적포구에 차례로 송신하는 토막에 번호를 붙이기때문에 그것들이 순서대로 도착하지 않으면 B의 TCP실체는 그것을 재배렬할수 있다.
- **검사합** : 송신 TCP는 토막의 나머지 부분의 내용에 대한 함수인 코드를 포함한다. 수신 TCP는 토막의 내용에 대하여 같은 계산을 진행하고 들어 오는 코드와 계산결과를 비교한다. 전송에서 어떤 오유가 발생했다면 불일치가 초래된다.

다음으로 TCP는 토막들을 B에로 전송하기 위하여 IP에 명령들과 함께 매개 토막을 넘겨 준다. 이 토막들은 한개 또는 그이상의 부분망들을 거쳐 전송되여야 하며 한개 또는 그이상의 중개경로기를 통하여 중계되여야 한다. 이 조작 역시 조종정보를 사용하여야 한다. 따라서 IP는 매개 토막에 조종정보머리부를 달아서 IP데이터그램을 만든다. IP머리부에 기억된 항목의 실례로는 목적가입자주소(이 실례에서는 B)를 들수 있다.

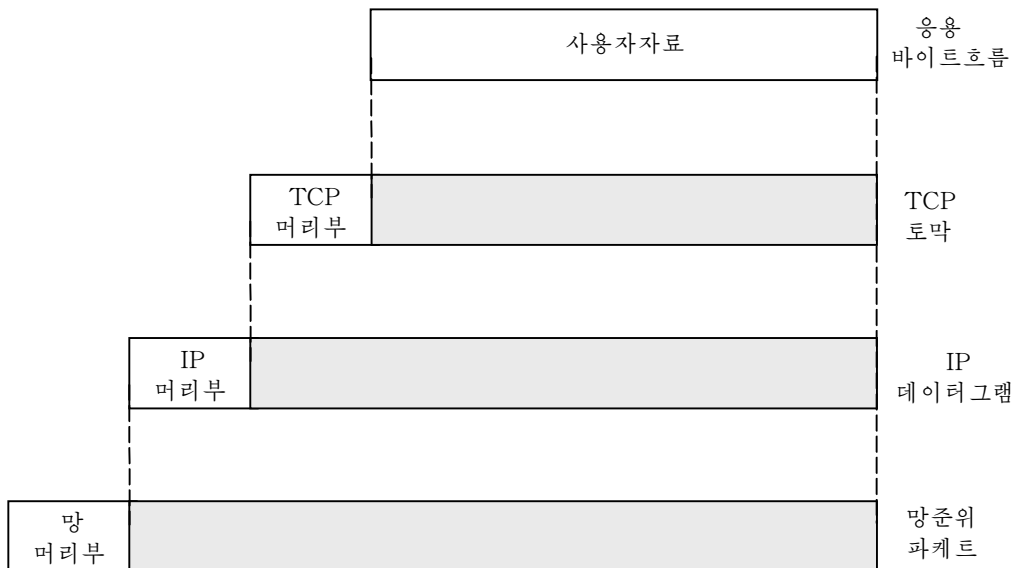



그림 부 1-2. TCP/IP방식에서 규약자료단위

마지막으로 매개 IP데이터그램은 목적지의 로정에 있는 첫 부분망을 통하여 전송하기 위하여 망접근층에로 넘겨 진다. 망접근층은 자기의 머리부를 달아 파케트 또는 프레임을 생성한다. 파케트는 부분망을 통하여 경로기 J에로 전송된다. 파케트머리부는 부분망이 부분망을 통하여 자료를 전송하는데 필요한 정보를 포함한다. 머리부에 포함시킬수 있는 항목의 실례로 다음과 같은것을 들수 있다.

- **목적부분망주소**: 부분망은 파케트가 어떤 련결장치로 전송되는가를 알아야 한다.
- **기능요청** : 망접근규약은 우선권과 같은 일정한 부분망기능들을 사용할것을 요청할수 있다.

경로기 J에서 파케트머리부는 제거되고 IP머리부가 조사된다. IP머리부에 있는 목적주소정보에 기초하여 경로기에서 IP모듈은 부분망 2를 통하여 B에로 데이터그램을 발송한다. 이를 위하여 데이터그램은 다시 망접근머리부와 련결된다.

자료가 B에서 수신되면 반대과정이 일어 난다. 매개 층에서 대응하는 머리부들이 제거되고 원본의 사용자자료가 목적프로쎄스에 도달할 때까지 나머지 부분은 보다 높은 층으로 넘겨 진다.

TCP와 UDP

TCP/IP규약구성방식의 부분으로 실행하는 대부분의 응용프로그람들에서 전송층규약은 TCP이다. TCP는 응용프로그람들사이의 자료전송을 위한 믿음직한 련결을 보장한다.

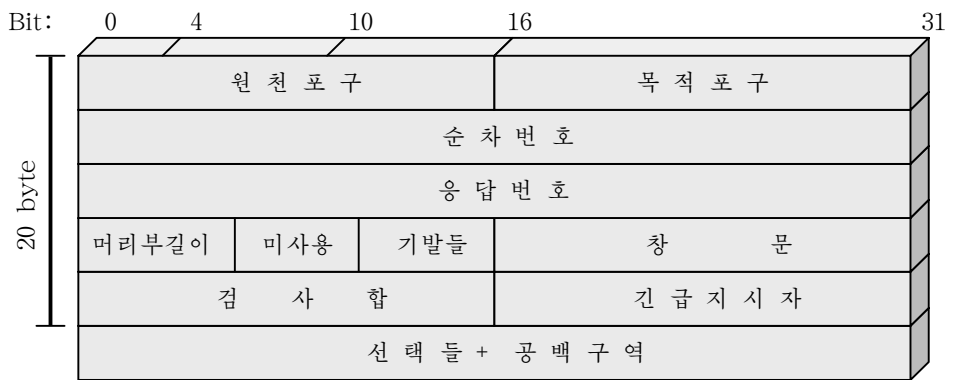
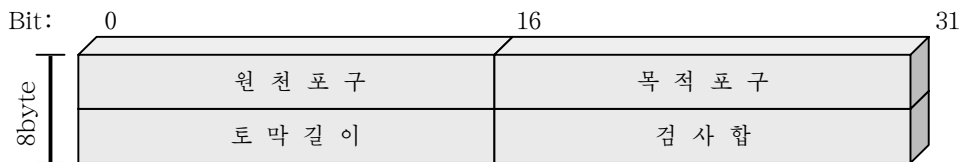



그림 부 1-3. TCP 와 UDP 머리부
ㄱ-TCP 머리부, ㄴ-UDP 머리부

그림 부1-3 ㄱ는 최소한 20byte 또는 160bit의 크기를 가지는 TCP의 머리부양식을 보여 준다. 원천포구와 목적포구마당들은 이 련결을 사용하고 있는 원천 및 목적체계들에서 응용프로그람들을 식별한다.[1] 순차번호, 응답번호, 창문마당들은 흐름조종과 오유조종을 제공한다. 본질상 매개 토막에는 루실토막을 검출할수 있도록 그리고 수신된 토막들에 대한 명백한 응답을 송신할수 있도록 번호가 매겨 진다. 매개 응답에 대하여 응답을 송신하는 실체는 얼마만한 추가적인 자료를 수신할 준비가 되여있는가를 창문마당으로 나타낸다. 검사합은 TCP토막에서 오유들을 검출하기 위하여 사용되는 16bit프레임검사배렬이다.

TCP외에 TCP/IP규약의 한 부분으로서 공통적으로 사용되고 있는 다른 전송준위규약인 사용자데이터그램규약(UDP)이 있다. UDP는 응용프로그람준위의 수속들에 무련결봉사를 제공한다. 그것은 전송, 순서의 보존, 복제의 보호를 보증하지 않는다. UDP는 수속들이 최소한의 규약기구로 다른 수속들에 통보문을 송신하도록 한다. 일부 트랜잭션지향응용프로그람들은 UDP를 사용한다. 한가지 실례로 TCP/IP망들의 표준망관리규약인 SNMP(간이망관리규약)을 들수 있다. UDP는 무련결방식이기때문에 UDP가 거의 할일이 없다. 본질상 그것은 IP에 포구주소지정능력을 첨가한것이다. 이것은 그림 부1-3 ㄴ에서 보여 준 UDP머리부를 보면 잘 알수 있다.

IP와 IPv6

몇십년동안 TCP/IP규약구성방식에서 기본은 망간통신규약(IP)이였다. 그림 부 1-4 ㄱ는 최소한 20byte 또는 160bit인 IP머리부형태를 보여 준다. 머리부는 32bit 원천 및 목적주소들을 포함한다. 머리부검사합마당은 전송오유를 피하기 위하여 머리부의 오유들을 검출하는데 사용된다. 규약마당은 TCP, UDP 또는 어떤 다른 보다 높은 층의 규약이 IP를 사용하고 있는가를 가리킨다. 기발들과 쪼각편차마당들은 단일 IP데이터그램을 전송하기 위하여 여러개의 IP데이터그램들로 나누고 목적지에서 다시 결합하는 쪼각화와 재결합처리에 사용된다.

1995년에 인터네트에 대한 규약규격들을 개발하는 인터네트공학과제집단(IETF)은 그 당시 IPng라는 다음세대 IP에 대한 규격을 발표하였다. 이 규격은 1996년에 IPv6이라는 규격과 교체되였다. IPv6(IPv4라는)은 현존 IP에 비하여 여러가지로 기능이 개선되여 오늘날 망의 보다 높은 속도와 더욱 우세해 지고 있는 도형과 영상을 비롯한 자료흐름들의 혼합을 처리할수 있도록 설계되였다. 그러나 새로운 규약을 개발하는 추동력은 더 많은 주소에 대한 요구였다. 현재 IP는 32bit주소를 사용하여 원천지 또는 목적지를 규정한다. 인터네트와 그에 결합된 구내망들이 독립적으로 장성함에 따라 이 주소길이로는 모든 체계들이 요구하는 주소를 수용하기가 불충분하게 되였다. 그림 부 1-4 ㄴ가 보여 주는바와 같이 IPv6은 128bit 원천 및 목적주소마당을 포함한다. 결국 TCP/IP를 사용하는 설비들을 현재의 IP에서 IPv6으로 이행할것이 기대된다. 그러나 이 과정은 몇십년 아니면 몇년 걸릴것이다.

TCP/IP응용

수많은 응용프로그람들이 TCP의 웃부분을 조작하기 위하여 표준화되였다. 여기서는 가장 공통적인것들중 세가지 응용프로그람만을 언급한다.

간이우편이송규약(SMTP)은 기본전자우편기능을 제공한다. 그것은 또한 분리되여

[1] 포구라는 용어는 대체로 OSI 관련문서들에서 사용되는 봉사접근점(SAP)이라는 용어에 대응한다.

있는 가입자들사이에서 통보문을 전송하기 위한 기구를 보장한다. SMTP의 특징으로는 우편목록기능, 왕복사용료기능, 발송기능을 들수 있다. SMTP규약은 통보문이 생성되는 방법을 규정하지 않는다. 다만 어떤 국부편집 또는 원시전자우편기능이 요구된다. 일단 통보문이 생성되면 SMTP는 통보문을 받으며 TCP를 사용하여 그것을 다른 가입자의 SMTP모듈에 송신한다. 목적 SMTP모듈은 국부전자우편제품을 사용하여 들어 오는 통보문을 사용자의 우편통에 보관할것이다.

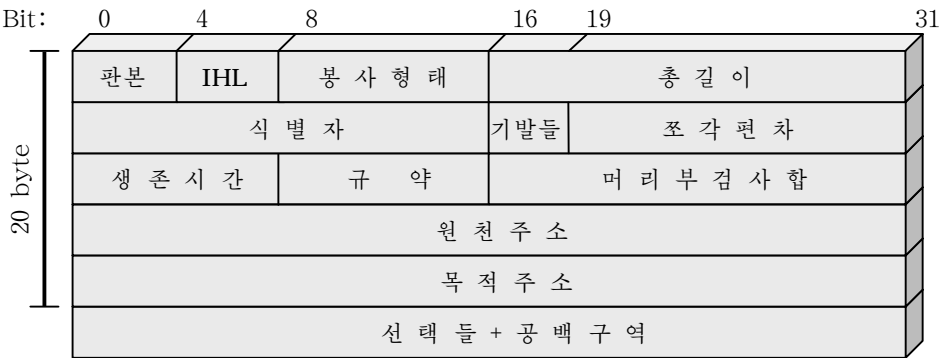
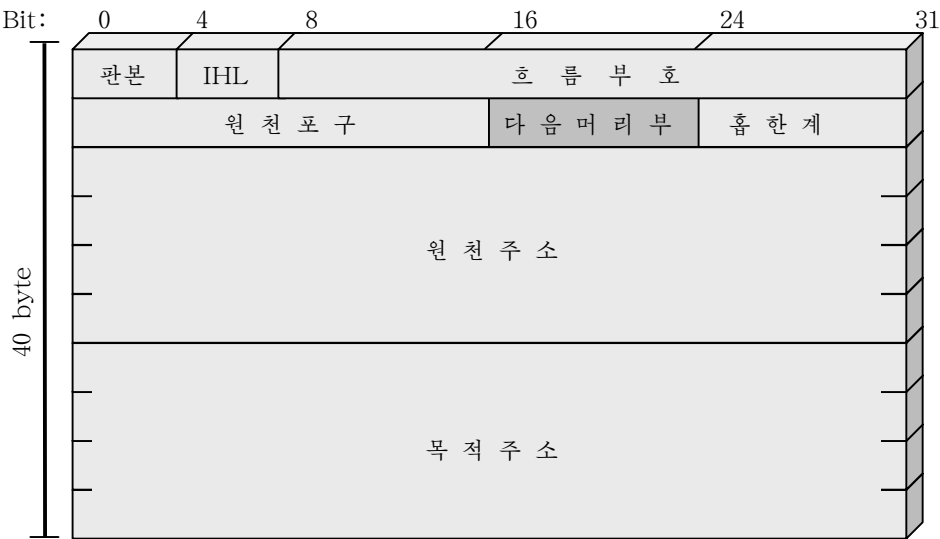
파일이송규약(FTP)은 사용자의 지령에 따라 한 체계에서 다른 체계에로 파일들을 송신하는데 사용된다. 본문과 2진파일을 다 전송할수 있으며 규약은 사용자의 접근을 조종하는 특징을 제공한다. 사용자가 파일전송을 시작하려고 할 때 FTP는 조종통보문들의 교환을 위하여 목적체계와 TCP련결을 설정한다. 이것은 사용자 ID와 통과암호가 전송되도록 하며 사용자는 파일과 필요한 파일조작들을 규정한다. 일단 파일전송이 허가되면 두번째 TCP련결이 자료전송을 위하여 설정된다. 자료는 응용프로그람수준에서 임의의 머리부들 또는 조종정보에 대한 내부처리가 없이 자료련결통로로 전송된다. 전송이 끝나면 조종련결을 사용하여 완성을 알리고 새로운 파일전송지령들을 접수한다.

ㄱ)

ㄴ)

그림 부 1-4. IP 머리부들
ㄱ-IPv4, ㄴ-IPv6

TELNET는 말단 또는 PC의 사용자가 원격콤퓨터에 직접 련결된것처럼 원격콤퓨터와 기능들에 가입하도록 하는 원격가입능력을 제공한다. 규약은 간단한 흘리기방식말단들과의 통신을 위하여 설계되였다. TELNET는 실제상 두개 모듈로 실현된다. 즉 사용자 TELNET는 국부말단과 통신하기 위하여 입출력모듈과 대화한다. 그것은 실제 말단들의 문자들을 망규격으로 그리고 그 반대로 변환한다. 봉사기 TELNET는 응용프로그람과 대화하여 원격말단이 응용프로그람에 국부적인것처럼 보이도록 대리말단조종기로서 동작한다. 사용자 TELNET와 봉사기 TELNET사이의 말단통화는 TCP련결통로에서 진행된다.

부록 2. 객체지향설계

Windows 2000과 최근 다른 여러가지 조작체계들은 객체지향설계원리들에 크게 의거하고 있다. 이 부록에서는 객제지향설계의 주요개념들을 간단히 개괄한다.

2-1. 동기

객체지향개념들은 교체할수 있는, 재사용할수 있는, 쉽게 갱신되며 쉽게 상호련결되는 쏘프트웨어부분품들에 대한 기대를 가지고 콤퓨터프로그람작성령역에서 매우 널리 보급되고 있다. 최근에는 자료기지설계가들이 객체지향자료기지관리체계들(OODMBS)이 출현하는데 따라 객체지향의 우점들을 평가하기 시작하였다. 조작체계설계자들은 또한 객체지향방법이 리득이 있다는것을 인식하였다.

객체지향프로그람작성과 객체지향자료기지관리체계는 사실상 다른것이지만 한가지 기본개념만은 공통으로 가지고 있다. 즉 쏘프트웨어 또는 자료를 《용기화》할수 있다는 것이다. 모든것은 통에 들어 가며 통안에는 통이 있을수 있다. 가장 간단한 전통적인 프로그람에서 한개 프로그람단계를 한개 명령과 같다고 본다. 객체지향언어에서 매개 단계는 통안에 꽉 찬 전체 명령일수 있다. 이와 류사하게 객체지향자료기지에서 한개 변수는 단일자료요소와 같다고 보지 않고 통안에 꽉 찬 전체 자료와 같다고 볼수 있다.

표 부 2-1은 객체지향설계에서 사용되는 몇가지 기본용어를 소개한다.

2-2. 객체지향의 개념

객체지향설계의 기본개념은 객체이다. 객체는 관련변수들(자료)과 성원함수들(수속들)의 집합을 포함하는 특수한 쏘프트웨어단위이다. 일반적으로 이 변수들과 성원함수들은 객체밖에서 직접 볼수 없다. 오히려 다른 사용자들이 자료와 수속들에 접근하도록 하는 잘 정의된 대면부들이 존재한다.

객체는 어떤 사물을 의미한다. 그것은 물리적실체, 개념, 쏘프트웨어모듈 또는 가상회선과 같은 어떤 동적실체이다. 객체에서 변수의 값들은 객체를 의미하는 사물에 대하여 알려 진 정보를 표현한다. 성원함수들은 그 집행이 객체의 변수들에 영향을 주며 역시 표현하고 있는 사물에 영향을 줄수도 있는 수속들을 포함한다.

그림 부2-1과 부2-2는 주요객체지향개념들을 설명한다. [1]

---

[1] 이 그림들은 Object International회사의 Peter Coad가 제출한 실례를 개작한것이다.

표 부 2-1. 주요객체지향용어들

용 어	정 의
속성	객체에 포함된 자료변수들
포함	포함하는 객체가 포함되는 객체의 지시자를 가지는 두개의 객체의 실체들사이의 관계
은폐	객체의 실체 속성들과 봉사들을 외부환경과 분리한것
계승	부모클라스의 속성들과 봉사들을 자식클라스가 획득하는 두 객체클라스들사이의 관계
통보문	객체들이 대화하는 수단
성원함수	객체의 부분이며 일정한 기능들을 수행하기 위하여 객체밖으로부터 동작시킬수 있는 수속
객체	실세계실체의 추상개념
객체클라스	같은 이름들과 속성들의 모임, 봉사들을 공유하는 이름을 가진 객체들의 모임
객체의 실체	속성들의 값이 할당된 객체클라스특정성원
다형성	봉사들에 대하여 같은 이름들을 사용하고 외부세계와 같은 대면부를 제공하지만 서로 다른 류형의 실체들을 표현하는 다중객체들의 존재를 말한다.
봉사	객체에 대한 조작을 수행하는 기능

객체의 구조

객체에 포함되는 자료와 수속들은 일반적으로 각각 변수, 성원함수라고 한다. 객체가《알고 있는》모든것은 객체의 변수로 표현될수 있으며 객체가 할수 있는 모든것은 그의 성원함수들로 표현될수 있다.

객체에서 **변수**(**속성**이라고도 함)들은 대표적으로 단순한 스칼라값들 또는 표들이다. 매개 변수는 형 또는 허용가능한 값들의 모임을 가질수 있으며 상수이거나 변수(편리상 변수라는 용어는 상수에도 적용된다)일수 있다. 접근제한은 일정한 사용자들, 사용자의 부류, 정황에 대한 변수들에 부여할수도 있다.

성원함수는 일정한 기능들을 수행하기 위하여 밖에서 시동될수 있는 수속들이다. 성원함수는 객체의 상태를 변경시키거나 일부 객체변수들을 갱신하거나 객체가 접근하는 외부자원들과 작용한다.

객체들은 **통보문**들을 통하여 대화한다. 통보문은 송신하는 객체의 이름, 수신하는 객체의 이름, 수신하는 객체에 있는 성원함수의 이름, 성원함수의 집행을 제한하는데 필요한 임의의 파라메터들을 포함한다. 통보문은 오직 객체안의 성원함수를 호출하는데만 사용될수 있다. 객체내부의 자료에 접근하는 유일한 방법은 객체의 성원함수에 의한 방법이다. 따라서 성원함수는 동작이 일어 나도록 할수 있거나 객체변수들에 접근하도록 할수 있다. 국부객체들에서 객체에로의 통보문넘기기는 객체의 성원함수의 호출과 같은것이다. 객체들이 분산될 때 통보문넘기기라고 하는것이 정확하다.

객체의 대면부는 객체가 지원하는 공개성원함수들의 모임이다. 대면부는 실현방법에 대하여 규정하지 않는다. 서로 다른 클라스들에서 객체들은 같은 대면부들에 대하여 서

로 다른 실현방법들을 가진다.

바깥세계와의 유일한 대면부가 통보문에 의거하는 객체의 성질을 **은페**라고 한다. 객체의 성원함수들과 변수들은 은페되며 오직 통보문에 기초한 통신을 통해서만 사용할수 있다. 이 성질은 두가지 우점을 가진다.

1. 그것은 객체의 변수들을 다른 객체들이 변경하지 못하게 보호한다. 이러한 보호에는 허용되지 않은 접근의 보호와 교착과 모순값들과 같이 병행접근으로 발생하는 문제들에 대한 보호가 속할수 있다.
2. 그것은 객체와의 대화가 상대적으로 간소하고 표준화되도록 객체의 내부구조를 숨긴다. 게다가 객체의 내부구조 또는 수속들이 외부기능을 변화시킴이 없이 변경된다면 다른 객체들은 영향을 받지 않는다.

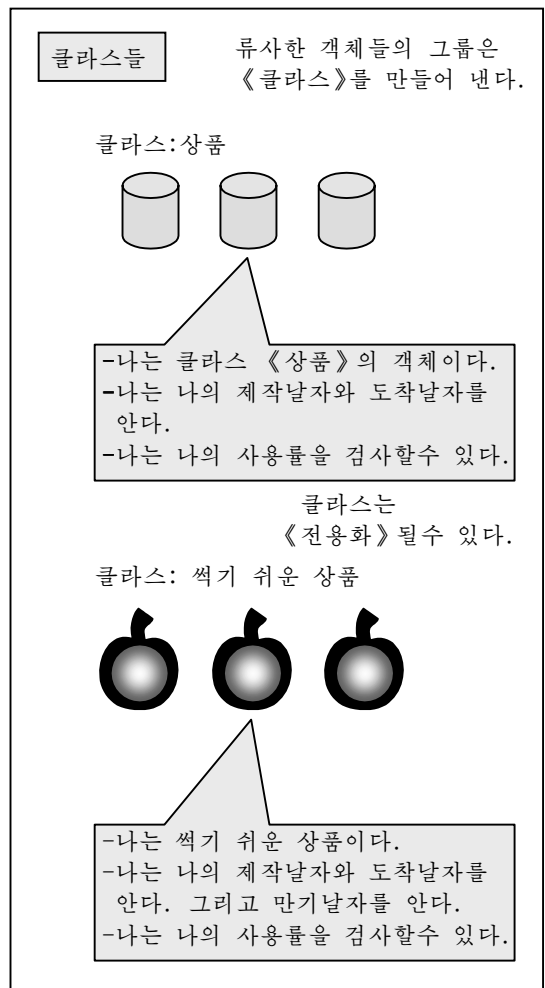



그림 부2-1. 객체들

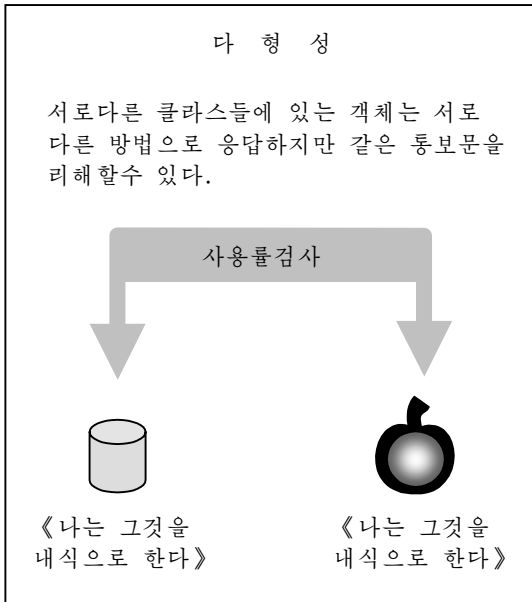
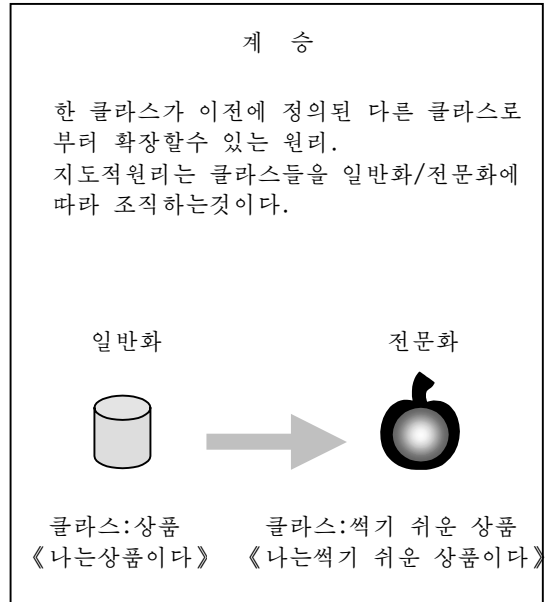
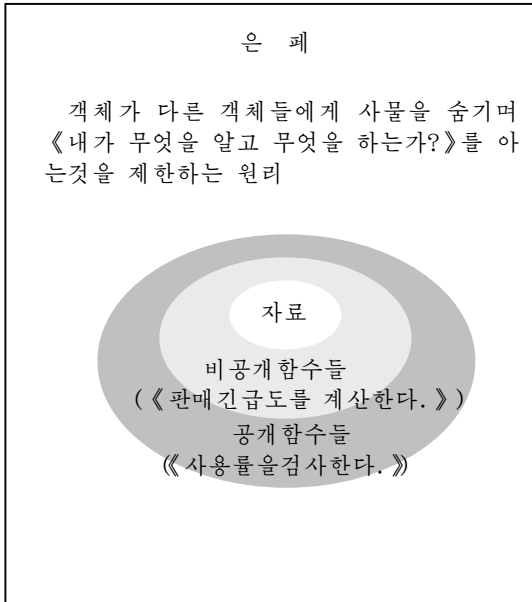



그림 부 2-2. 객체개념들

객체클라스

실천에는 같은 류형의 사물을 표현하는 수많은 객체들이 있다. 실례로 프로쎄스를 객체로 표현한다면 그때에는 체계에 존재하는 매개 프로쎄스에 하나의 객체가 존재할것이다. 명백히 이러한 매개 객체는 자기의 변수모임을 필요로 한다. 객체의 성원함수들이 재진입가능한수속들이라면 모든 류사한 객체들은 같은 성원함수들을 공유할수 있다. 게

다가 새로우면서 류사한 객체마다 성원함수들과 변수들을 재정의하는것은 비능률적이다.

이러한 난관을 해결하는 방도는 객체클라스와 객체구체례를 구별하는것이다. **객체클라스**는 특정한 형의 객체에 포함되는 성원함수들과 변수들을 정의하는 본보기이다. **객체구체례**는 자기를 정의하는 클라스의 특징들을 포함하는 실제객체이다. 구체례는 객체클라스에서 정의된 변수들의 값들을 포함한다.

계승

객체클라스는 최소한의 공수로 많은 객체구체례들을 생성할수 있도록 하기때문에 객체클라스의 개념은 매우 위력하다. 이 개념은 계승기구를 사용하여 더 강력해 진다 [TAIV96].

계승은 새로운 객체클라스를 현존클라스로 정의하도록 한다. **파생클라스**라고 하는 새로운 (보다 낮은 수준의) 클라스는 자동적으로 **프로젝트**라는 원래의(보다 높은 수준의) 클라스의 성원함수들과 변수정의들을 포함한다. 파생클라스는 몇가지 점에서 상위클라스와 다를수 있다.

1. 파생클라스는 상위클라스에 없는 추가적인 성원함수들과 변수들을 포함할수 있다.
2. 파생클라스는 새로운 정의에 같은 이름을 사용하여 상위클라스에 있는 임의의 성원함수 또는 변수를 재정의할수 있다. 이것은 특정한 실례들을 조절하기 위한 단순하고 효률적인 방법을 준다.
3. 파생클라스는 어떤 방법으로 상위클라스에서 계승한 성원함수 또는 변수를 제한할수 있다.

계승기구는 재귀적이며 파생클라스가 자기 파생클라스의 상위클라스로 되게 한다. 이런 식으로 계승계층이 구축될수 있다. 개념적으로 성원함수와 변수들에 대한 탐색수법을 정의하는것으로 계승계층을 생각할수 있다. 객체가 자기 클라스에 정의되지 않은 성원함수를 찾아서 수행할데 대한 통보문을 수신하면 자동적으로 계승계층을 탐색하여 그 성원함수를 찾는다. 이와 류사하게 성원함수의 집행이 클라스에서 정의되지 않은 변수들에 대한 참조를 초래한다면 객체는 계승계층에서 변수이름을 탐색한다.

다형성

다형성은 공통대면부에 서로 다른 실현물들을 숨길수 있는 강력하고 흥미를 끄는 특징이다. 서로 다형인 두 객체는 성원함수에 같은 이름을 사용하며 다른 객체들에 같은 대면부를 준다. 실례로 점행렬인쇄, 레이자인쇄, 화면인쇄 기타 등등과 같은 서로 다른 출구장치들에 대한 수많은 인쇄객체들이 존재할수 있다. 또는 본문인쇄, 도안인쇄, 부분품인쇄와 같은 서로 다른 형의 문서들에 대한 수많은 인쇄객체들이 존재할수 있다. 이렇게 매개 객체가 인쇄라고 하는 성원함수를 가진다면 성원함수가 실제로 어떻게 수행되는가에는 관계없이 적당한 객체에로 통보문인쇄를 송신하여 임의의 문서를 인쇄할수 있다.

다형성과 쓸모 있는 모듈프로그람작성수법들을 비교하는것이 리해에 도움이 된다. 모듈식내리설계의 목적은 보다 높은 수준의 모듈과의 고정된 대면부를 가지는 보다 낮은 수준의 일반편의프로그람의 모듈들을 설계하는것이다. 이것은 보다 낮은 수준의 한개 모듈이 서로 다른 보다 높은 수준의 많은 모듈들에 의하여 기동되게 할수 있다. 보다 낮은 수준의 모듈의 내부구조들이 그것의 대면부를 변경시킴이 없이 변경된다면 그것을 사용하는 보다 높은 준위의 모듈은 전혀 영향을 받지 않는다. 그와는 대조적으로 다형성에서는 한개의 보다 높은 수준의 객체에 의하여 류사한 기능들을 수행하도록 같은 통보문형식을 사용하는 서로 다른 보다 낮은 수준의 많은 객체들을 기동하기 위한 능력이 문제로

된다. 다형성의 경우에 현존객체들에 대한 최소한의 변경으로 새로운 보다 낮은 수준의 객체들을 추가할수 있다.

포함

다른 객체들을 포함하는 객체구체례들을 **합성객체**들이라고 한다. 포함은 한 객체에로의 지시자를 다른 객체에서 값으로 포함하는것으로 실현할수 있다. 합성객체들의 우점은 그것들이 복잡한 구조들을 표현할수 있게 한다는것이다. 실례로 합성객체에 포함된 객체는 그자체가 합성객체일수 있다.

대표적으로 합성객체들로 구축된 구조들은 나무위상구조로 제한된다. 즉 순환참조는 전혀 허용되지 않으며 매개 《자식》객체구체례는 오직 한개의 《부모》객체구체례를 가질수 있다.

객체클라스의 계승계층과 객체구체례들의 포함계층과의 차이를 명백히 하는것이 중요하다. 이 두개는 서로 관계가 없다. 계승을 사용하여 최소한의 노력으로 많은 서로 다른 객체류형들을 정의할수 있다. 포함을 사용하여 복잡한 자료구조들을 구축할수 있다.

2-3. 객체지향설계의 우점

[CAST92]는 객체지향설계의 우점을 다음과 같이 제시한다.

- **천성적복잡성의 합리적인 조직** : 계승을 사용하여 관련개념들과 자원들 그리고 다른 객체들을 능률적으로 정의할수 있다. 포함을 사용하여 기초를 이루는 과제를 즉시에 반영하는 임의의 자료구조들을 구축할수 있다. 객체지향프로그람작성언어들과 자료구조들에 의하여 설계자들은 조작체계자원들과 기능들에 대한 자기들의 리해를 반영하는 방법으로 이 자원들과 기능들을 서술할수 있다.
- **재사용에 의한 개발공수의 감소** : 다른 사람들이 작성하고 검사 및 보수한 객체클라스들을 재사용하면 개발, 검사, 보수시간을 줄일수 있다.
- **더 확장가능하고 보수가능한 체계** : 제품개선과 수리들을 비롯한 보수는 전통적으로 임의의 제품수명주기원가의 약 65%를 소비한다. 객체지향설계는 그 프로수를 낮춘다. 객체에 기초한 쏘프트웨어의 사용은 서로 다른 쏘프트웨어부분들의 가능한 상호작용의 수를 제한하도록 하여 체계의 나머지부분에 영향을 적게 주도록 클라스에 대한 변경을 실현할수 있다.

이 우점들은 조작체계를 객체지향체계로 설계하도록 추진하고 있다. 객체들로 하여 프로그람작성자들은 체계통합을 파괴함이 없이 조작체계를 전용화하여 새로운 요구들을 만족시킬수 있다. 객체들은 또한 분산계산을 가능하게 한다. 객체들은 통보문들로 통신하기때문에 두개의 통신객체들이 같은 체계상에 있든 망의 두개의 서로 다른 체계상에 있든 관계없다. 자료, 함수들과 스레드들은 필요에 따라 워크스테이션들과 봉사기들에 동적으로 할당된다. 따라서 조작체계설계를 위한 객체지향방법은 PC와 워크스테이션조작체계들에서 점차 명백해 지고 있다.

2-4. CORBA

이 책에서 이미 본바와 같이 객체지향개념은 조작체계핵심부설계와 실현물에 사용되여 유연성과 조작성, 이식성에서 리익을 가져 왔다. 객체지향수법들의 사용으로 인한 리익은 분산조작체계들을 비롯한 분산쏘프트웨어령역에로 동등하게 또는 그이상으로 확장

된다. 분산쏘프트웨어의 설계와 실현을 위한 객체지향수법들의 응용을 분산객체계산(DOC)이라고 한다.

DOC방법이 나오게 된 동기는 분산쏘프트웨어를 만드는데서 난관이 증가하였기때문이다. 즉 계산 및 망하드웨어가 더 작아 지고 더 빨라 지며 더 눅어 지는 사이에 분산쏘프트웨어는 더 커지고 더 느려 지고 개발과 보수가 더 비싸 진다. [SCHM97]은 분산쏘프트웨어에 대한 도전이 두가지 복잡성으로부터 유래된다고 지적한다.

- **천성적복잡성** : 이것은 분산의 기초적인 문제들로부터 발생한다. 이 문제들중에서 주요한것은 망과 가입자의 고장을 검출하고 회복하며 통신처리시간의 영향을 최소로 하고 망의 콤퓨터들에 있는 봉사부분품들과 작업부하의 최적분할을 결정하는것이다. 더우기 자원페색과 교착문제가 있는 병행프로그람처리는 여전히 어려우며 분산체계들은 천성적으로 병행적이다.
- **우연한 복잡성** : 이것은 분산쏘프트웨어의 구축에 필요한 도구들과 수법들을 제한하는데로부터 발생한다. 우연한 복잡성의 공통원천지는 확장불가능 및 재사용불가능체계들을 초래하는 기능적설계를 널리 사용하는데 있다.

DOC는 두가지 형태의 복잡성을 처리하기 위한 믿음직한 방법이다. DOC방법에서 가장 중요한것은 국부객체와 원격객체들사이의 통신중개자로서 동작하는 객체요청중개자들(ORB들)이다. ORB들은 분산응용프로그람들을 설계하고 실현하는데서 지루하고 오유가 발생하기 쉬우며 간편하지 않은 측면들을 제거한다. ORB를 추가하자면 응용프로그람들과 객체지향하부구조사이의 대면부정의와 통보문교환을 위한 수많은 규약들과 양식들이 있어야 한다.

DOC시장에서 경쟁하는 기본수법에는 세가지가 있다. 그것은 공통객체요청중개자구성방식(CORBA)이라고 하는 객체관리그룹(OMG)방식, JAVA원격성원함수호출(RMI)체계, Microsoft의 분산부분품객체모형(DCOM)이다. CORBA는 세가지 수법가운데서 가장 발전되고 잘 확립된 수법이다. IBM, Sun, Netscape, Oracle을 비롯한 수많은 발전된 공업회사들은 CORBA를 지원하며 Microsoft는 자기의 Windows에 있는 DCOM을 CORBA와 련결할것이라고 공포하였다. 이 부록의 나머지 부분에서 CORBA를 간단히 개괄한다.

표 부2-2는 CORBA에서 사용되는 몇가지 기본용어들을 제시한다. CORBA의 기본특징은 다음과 같다(그림 부2-3).

- **의뢰기** : 의뢰기들은 기초를 이루는 ORB가 제공하는 여러가지 기구를 통하여 요청들과 접근객체봉사들을 발생한다.
- **객체실현물** : 이것들은 분산체계에 있는 여러가지 의뢰기들이 요청하는 봉사들을 제공한다. CORBA방식의 한가지 우점은 의뢰기들과 객체실현물들을 많은 프로그람작성언어들로 작성할수 있으며 여전히 충분한 령역의 요청봉사들을 제공할수 있다는것이다.
- **ORB핵심** : ORB핵심은 객체들사이의 통신을 담당한다. ORB는 망의 객체를 찾고 객체에로 요청들을 배달하며 객체들을 유효하게 한다(만일 유효하지 않다면). 그리고 송신자에게 임의의 통보문을 넘겨 준다. ORB핵심은 프로그람작성자들이 국부성원함수 또는 원격성원함수를 기동할 때 같은 파라메터들을 가진 같은 성원함수를 정확히 사용하기때문에 **접근투명성**을 제공한다. ORB핵심은 또한 **위치투명성**을 제공한다. 즉 파라메터들은 객체의 위치를 규정할것을 요구하지 않는다.

표 부 2-2. 분산CORBA체계의 기본개념들

CORBA개념	정 의
의뢰기응용프로그람	객체에 대한 조작들을 수행하기 위하여 봉사기에 요청들을 호출한다. 의뢰기응용프로그람은 의뢰기가 요청할수 있는 객체들과 조작들을 서술하는 한개 또는 그이상의 대면부정의들을 사용한다. 의뢰기응용프로그람은 요청을 위하여 객체들이 아니라 객체참조들을 사용한다.
례외정보	요청이 성과적으로 수행되였는가를 가리키는 정보를 포함한다.
실현물	객체조작과 관련된 작업을 수행하는 한개 또는 그이상의 성원함수들을 정의하고 포함한다. 봉사기는 한개 또는 그이상의 실현물들을 가질수 있다.
대면부	어떤 조작이 그 객체들에 유효한가 하는것과 같이 객체의 구체례들이 어떻게 행동하는가를 서술한다.
대면부정의	일정한 류형의 객체에 유효한 조작들을 서술한다.
기동	요청을 송신하는 프로쎄스
성원함수	조작과 관련된 작업을 수행하는 봉사기코드. 성원함수는 실현물들에 포함된다.
객체	사람, 장소, 사물 또는 쏘프트웨어부분을 표현한다. 객체는 종업원객체에 대한 촉진조작과 같이 자기에 대하여 수행되는 조작들을 가질수 있다.
객체구체례	한개의 특정한 종류의 객체의 사건
객체참조	객체구체례의 식별자
OMG대면부정의	CORBA에서 대면부들을 정의하기 위한 정의언어(IDL)
조작	의뢰기가 봉사기에 요청하여 객체구체례에 대하여 수행할수 있는 작용
요청	의뢰기와 봉사기응용프로그람사이에서 송신되는 통보문
봉사기응용프로그람	객체들과 그것들에 대한 조작들의 한개 또는 그이상의 실현물들을 포함한다.

- **ORB핵심** : ORB핵심은 객체들사이의 통신을 담당한다. ORB는 망의 객체를 찾고 객체에로 요청들을 배달하며 객체들을 유효하게 한다(만일 유효하지 않다면). 그리고 송신자에게 임의의 통보문을 넘겨 준다. ORB핵심은 프로그람작성자들이 국부성원함수 또는 원격성원함수를 기동할 때 같은 파라메터들을 가진 같은 성원함수를 정확히 사용하기때문에 **접근투명성**을 제공한다. ORB핵심은 또한 **위치투명성**을 제공한다. 즉 파라메터들은 객체의 위치를 규정할것을 요구하지 않는다.
- **대면부** : 객체의 대면부는 객체가 지원하는 조작들과 류형들을 규정하며 따라서 객체에 대한 요청들을 정의한다. CORBA대면부들은 C++의 클라스들과 JAVA의 대면부들과 류사하다. C++클라스들과는 달리 CORBA대면부는 성원함수들과 그것들의 파라메터들 그리고 복귀값들을 규정하지만 그것들의 실현물에 대해서는 규정하지 않는다. 같은 C++클라스의 두개의 객체는 자기 성원함수들의 같은 실현물을 가진다.

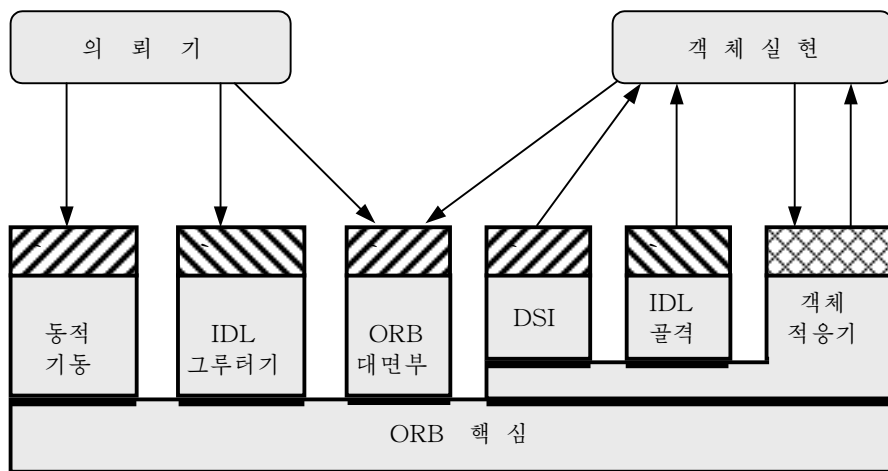



그림 부 2-3. 공통객체요청중개자구성방식

- **OMG대면부정의언어(IDL)** : IDL은 객체들을 정의하는데 사용되는 언어이다. IDL대면부정의의 실례는 아래와 같다.

```
//OMG IDL
                    interface Factory
{
 Object create( );
};
```

이 정의는 한개의 조작 create를 지원하는 Factory라는 대면부를 규정한다. create 조작은 파라메터를 전혀 가지지 않으며 객체참조형 object를 넘겨 준다. 객체형Factory에 대한 객체참조가 주어 지면 의뢰기는 새로운 CORBA객체를 생성하기 위하여 그것을 기동할수 있다. IDL은 프로그람작성독립언어이며 이로 하여 의뢰기는 직접 임의의 조작을 기동하지 못한다. 그것을 수행하기 위하여 IDL을 의뢰기프로그람작성언어로 넘겨야 한다. 또한 서로 다른 프로그람작성언어들로 봉사기와 의뢰기의 프로그람을 작성할수 있다. 규격언어를 사용하면 여러개의 언어들과 가동환경들에서 서로 다른 처리를 취급할수 있다. 따라서 IDL은 **가동환경독립**을 가능하게 한다.

- **언어맺기창조** : IDL콤파일러들은 한개의 OMG IDL파일을 Ada, C, C++, COBOL 등과 같은 객체지향이거나 객체지향이 아닌 서로 다른 프로그람작성언어들로 사

영한다. 이 사영은 언어에 특유한 자료형들과 봉사객체들에 접근하기 위한 수속대면부들, IDL의뢰기그루터기대면부, IDL골격, 객체적응기들, 동적골격대면부, 직접 ORB대면부들의 정의를 동반한다. 보통 의뢰기들은 객체대면부의 콤파일-시간지식을 가지며 의뢰기그루터기들을 사용하여 정적기동를 한다. 어떤 경우에 의뢰기들은 이러한 지식을 가지지 않으며 동적기동을 한다.

- **IDL그루터기 :** 의뢰기응용프로그람을 위하여 ORB핵심부에 호출한다. IDL그루터기들은 ORB핵심기능들을 종단-의뢰기응용프로그람들이 채용할수 있는 직접 RPC(원격수속호출)기구들로 요약하는 기구들의 모임을 제공한다. 이 그루터기들은 ORB와 원격객체실현물의 결합을 마치 그것들이 동일한 즉시처리프로쎄스에 련결되여 있는것처럼 생각되게 한다. 대부분의 경우에 IDL콤파일러들은 의뢰기와 객체실현물들사이의 대면부를 완성하는 언어특유의 대면부서고들을 발생한다.
- **IDL골격 :** 이것은 특정봉사성원함수들을 기동하는 코드를 제공한다. 정적 IDL골격들은 의뢰기측 IDL그루터기들에 대한 봉사기측보충물들이다. 그것들은 의뢰기와 객체실현물들사이의 련결을 완성하는 객체실현물들과 ORB핵심부사이의 맺기들을 포함한다.
- **동적기동 :** 동적기동대면부(DII)를 사용하면 의뢰기응용프로그람은 객체의 대면부들에 대한 콤파일-시간지식이 없이도 임의의 객체에 대한 요청을 기동할수 있다. 대면부세부들은 대면부서고 그리고 다른 실행시원천들을 참고하여 작성된다. DII에 의하여 의뢰기는 응답이 전혀 없는 한방향지령들을 출구한다.
- **동적골격대면부(DSI) :** IDL그루터기들과 IDL골격들과의 관계와 마찬가지로 DSI는 객체들에로의 동적배분을 보장한다. 봉사기측에 대한 동적기동과 동일하다.
- **객체적응기 :** 객체들의 활성화와 실현물들의 활성화와 같이 일반 ORB관련과제들을 조절하기 위하여 CORBA제작자가 제공하는 CORBA체계부분품이다. 적응기는 이 일반과제들을 가지고 있으며 그것들을 봉사기의 특정실현물들과 성원함수들에 결합한다.

부록 3. 프로그람작성 및 조작체계의 프로젝트

많은 강사들은 실현물 또는 연구프로젝트들이 조작체계개념들을 명백하게 리해하는데서 결정적이라고 믿고 있다. 프로젝트가 없이는 학생들이 기초적인 OS추상개념들과 부분들사이의 호상작용에 대해 파악하기가 어렵다. 많은 학생들이 파악하기가 힘들어 하는 개념의 좋은 실례로 신호기를 들수 있다. 프로젝트는 이 책에서 소개된 개념들을 보강하며 학생들에게 OS의 서로 다른 부분들이 어떻게 잘 조화되는가에 대한 더 큰 리해를 주며 그들에게 OS의 세부를 리해할뿐아니라 실현할수 있는 능력을 준다.

이 책에서는 가능한껏 명백하게 OS내부의 개념들을 주기 위해 노력하였고 그 개념을 보충하기 위해 약 170개 숙제문제를 주었다. 그러나 많은 강사들이 프로젝트로 이 문제를 보충하려고 할것이다. 이 부록은 그러한 측면에서 몇가지 지침을 주며 강사지도서에서 사용할수 있는 문제들을 서술하였다. 더 구체적인것은 부록 4와 5에서 주었다.

3-1. 교육용조작체계의 프로젝트

강사는 다음의 방법들에서 한가지 방법을 선택할수 있다.

- **조작체계프로젝트(OSP) :** OSP는 현대 조작체계의 실현물이며 OS설계에 대한 기초강의에 적합한 실현프로젝트를 산생하기 위한 유연한 환경이다. OSP는 많은

프로젝트들을 동반한다.

- **Ben-Ari병렬해석기(BACI)** : BACI는 병렬프로쎄스집행을 모의하며 2진 및 계수신호기들과 감시기들을 지원한다. BACI는 많은 프로젝트들과 동반되여 병렬개념을 보충하는데 사용된다.
- **Nachos**: OSP의 경우와 마찬가지로 Nachos는 개념을 보충하기 위해 실현물프로젝트들을 산생하기 위한 환경이며 역시 많은 프로젝트들을 동반한다.
- **연구프로젝트** : 강사들의 지도서는 대학생들이 인터네트상의 특정한 문제를 연구하고 보고서를 작성하는것과 같은 대학생들에게 줄 연구과제를 제시한다.
- **프로그람작성프로젝트** : 강사들의 지도서는 책에서의 개념을 보충하기 위하여 과제로 줄수 있는 작은 프로그람프로젝트들의 모임을 준다. 과제수행을 위하여 모든 언어를 사용할수 있다. 프로젝트는 이 책에서 작성하는 넓은 범위의 화제를 포괄한다.
- **읽기/보고서과제** : 강사들의 지도서는 대학생들이 론문을 분석하는 간단한 보고서를 제출하도록 과제를 줄수 있는 매개 장에 한개 또는 그이상의 론문들로 구성되는 중요한 론문들의 목록을 포함한다.

이 부록은 이상의 문제들을 간단히 론의한다. 부록 4는 OSP에 대한 구체화된 소개와 함께 체계와 프로그람작성과제를 얻는 방법에 대한 정보를 제공한다. 부록 5는 BACI에 대한 정보를 제공한다. Nachos는 Web싸이트에서 충분한 자료가 제공되고 있으며 다음 절에서 간단히 서술한다.

3-2. Nachos

Nachos 의 개괄

Nachos는 대학생들에게 재생가능한 착오수정환경을 제공하기 위하여 UNIX프로쎄스로서 실행하며 OS와 그의 밑준위에 있는 하드웨어를 모의하는 교육용조작체계이다. 목적은 실제 OS가 어떻게 작업하는가를 보여 주는데 충분히 현실적이면서도 대학생들이 의의 있는 방법으로 리해하고 수정하기에 충분히 간단한 프로젝트환경을 보장하는것이다.

자유배포제품은 다음과 같은것들과 함께 Web를 통하여 입수할수 있다.

- 개괄론문
- 작업조작체계를 위한 간단한 기저선코드
- 일반적인 개인콤퓨터/워크스테이션을 위한 모의기
- 표본과제:과제는 스레드와 병행성, 다중프로그람처리, 체계호출, 가상기억기, 쏘프트웨어에 의하여 적재된 TLB들, 파일체계, 망통신규약, 원격수속호출과 분산체계들을 비롯한 현대 OS의 모든 령역을 설명하고 조사한다.
- C++초보(Nachos는 배우기 쉬운 C++부분모임으로 작성되였으며 초보는 C프로그람작성자들에게 이 부분모임을 배워 준다.)

Nachos는 세계의 수백개 대학들에서 사용되여 왔으며 LINUX, Free BSD, Net BSD, DEC MIPS, DEC Alpha, Sun Solaris, SGE IRIX, HP-UX, IBM AIX, MS-DOS, Apple Macintosh를 비롯한 수많은 체계에 도입되였다. 앞으로는 SGE워크스테이션의 완전한 기계모의인 Stanford의 Sim OS의 포구를 포함할것이다.

Nachos는 그것의 Web싸이트로부터 마음대로 입수할수 있다(William Stalling com/OS4e, htme로부터 그 Web싸이트에 련결한다.). 해결방법들은 hachos @ cs,

berkeley, edu로부터 e-mail에 의해 강사들이 입수할수 있다. 게다가 강사와 새소식그룹(alt, os, nachos)을 위한 우편목록이 있다.

Nachos, OSP와 BACI의 선택

강사가 대학생들이 사용할수 있는 국부환경에 이 세가지 모의기중 한가지 모의기를 도입하려고 할 때 이 세가지중 어느것을 선택하는가는 강사의 목적과 개인의 의견에 따른다. 프로젝트의 초점이 병행성에 있다면 명백히 BACI를 선택한다. BACI는 신호기, 감시기, 병행프로그람작성의 복잡성과 난문제들을 연구하기 위한 우월한 환경을 보장한다.

대신에 강사 또는 대학생들이 병행프로그람처리, 주소공간과 일정작성, 가상기억기, 파일체계, 망 기타 등을 비롯하여 여러가지 OS기구들을 연구하고 싶다면 Nachos나 OSP를 사용할수 있다.

OSP가 OS프로젝트들을 지원하는데 사용할수 있는 가장 좋은 매개물들중의 하나라고 생각되기때문에 부록에 포함시켰다. OSP는 100개이상의 싸이트들에서 사용되고 있으며 많은 자원프로그람의 상세한 자료를 제공한다. 한가지 잠재적인 약점은 비록 체계와 표본과제들, 우편목록지원들이 무료라고 하여도 대학생들이 작은 사용자안내서를 구입하여야 한다는것이다. 그러나 이것은 이 환경의 효과들과 비교되여야 한다. Nachos는 OSP와 마찬가지로 널리 사용되고 있으며 지원, 문서, 제안연구과제들을 준다. 강사들이 부록 4를 연구할 필요가 있으며 관심이 있다면 그것을 Nachos의 Web싸이트에서 입수할수 있는 Nachos개괄론문과 다른 자료와 비교해 보시오.

3-3. 연구프로젝트

강의에서 배운 기본개념들을 보충하고 대학생들에게 연구수법을 배워 주기 위한 효과적인 방법은 연구프로젝트를 배당하는것이다. 이러한 프로젝트는 제작자제품들과 연구실활동, 표준화안들에 대한 Web조사는 물론 문헌조사를 동반할수 있다. 팀에는 프로젝트들을 배당하고 개인에게는 보다 작은 프로젝트들을 배당할수 있다. 어떤 경우이든 학기초기에 어떤 종류의 프로젝트제안을 요구하여 제안을 적당한 화제와 적당한 수준의 공수로 평가하기 위한 강사시간을 주는것이 가장 합리적이다. 대학생들에게 주는 연구프로젝트를 위한 배포인쇄물은 다음과 같은것을 포함한다.

- 제안을 위한 양식
- 최종보고서를 위한 양식
- 중간 및 최종기한부일정작성
- 가능한 프로젝트화제들의 목록

대학생들은 제시된 문제들중 하나를 선택하거나 비교될만한 프로젝트들을 자체로 고안해 낼수 있다. 강사의 지도서는 George Mason종합대학의 Tan N.Nguyen교수가 개발한 가능한 연구목록은 물론 제안 및 최종보고서를 위한 제안양식을 포함한다.

3-4. 프로그람작성프로젝트

OSP 또는 Nachos를 사용하거나 병행성에 초점을 두고 BACI를 사용해서 OS의 부분들을 개발하기 위한 다른방법은 하부구조를 전혀 필요로 하지 않는 프로그람작성프로젝트들을 많이 배당하는것이다. OSP나 BACI와 같은 자원들을 사용하는데 비하여 프로그람작성프로젝트들은 여러가지 우점을 가진다.

1. 강사는 프로젝트를 배당하기 위하여 지원틀과 일치하는 개념들이 아니라 OS와 관련되는 여러가지 다양한 개념들을 선택할수 있다.
2. 프로젝트들은 임의의 사용가능한 콤퓨터에서 임의의 적당한 언어로 대학생들에 의하여 프로그람작성될수 있다.
3. 강사는 하부구조를 내리적재, 설치, 구성할 필요가 없다.

프로젝트의 크기는 임의로 할수 있다. 보다 큰 프로젝트들은 높은 학력을 가진 대학생에게는 적합하지만 능력이나 조직적수법이 충분하지 못한 대학생들은 할수 없다. 큰 프로젝트들은 보통 가장 우수한 대학생들로 하여금 자기들의 능력을 충분히 발휘할수 있게 한다. 작은 프로젝트일수록 개념들로부터 프로그람작성을 보다 쉽게 할수 있으며 그것들중에서 대부분을 과제로 줄수 있기때문에 여러가지 다른 문제들을 강조할 기회를 준다. 여러가지를 고려하여 보다 작은 프로젝트들을 취급하는것이 합리적이다. 따라서 강사의 지도서에는 일련의 작은 프로젝트들을 포함되여 있다. 매개 프로젝트는 한주일정도로 완성될수 있으며 대학생과 교원에게 다 만족될수 있다. 이 프로젝트들은 Worcester Polytechic협회의 Stephen Taylor가 개발하였다. 조작체계를 가르치는 강의에서 그 프로젝트들을 수십번 사용하고 재현하였다.

3-5. 문헌연구/보고서의 작성과제

강의에서 배운 개념들을 보충하고 대학생들에게 연구경험을 주기 위한 다른 우수한 방법은 읽고 분석할 문헌의 론문들을 배당하는것이다. 강사지도서는 매개 장에 하나 또는 그이상씩 배당될 제안론문목록을 포함한다. 모든 론문들은 인터네트나 임의의 훌륭한 대학기술도서관을 통하여 쉽게 입수할수 있다. 지도서는 역시 제안된 과제의 용어를 포함한다.

부록 4. OSP:조작체계프로젝트의 환경

4-1. 개괄

OSP는 현대 조작체계의 실현물이며 조작체계설계에 대한 초보강의에 적합한 실현물프로젝트들을 발생하기 위한 유연한 환경이다. OSP는 조작체계들에 대한 초보교과서의 사용을 보충하려고 계획되였으며 3학기동안 취급하는데 충분한 프로젝트들을 포함한다. 이 프로젝트들은 대학생들이 조작체계의 많은 본질적인 특징들에 접하게 하는 동시에 저준위기계의존사건들과 분리시킨다. 따라서 대학생들은 한 학기동안에 가상기억기관리에서의 페지교체, 처리기일정작성전략, 디스크탐색시간최적화와 조작체계설계에서 제기되는 다른 문제들을 배울수 있다.

OSP는 C프로그람작성언어로 작성되며 대학생들은 C언어로도 자기의 OSP프로젝트들을 작성할수 있다. 그러므로 OSP를 사용하기 위한 필요조건으로서 대학생들은 기초가 든든한 C프로그람작성기교를 가져야 하며 자료구조들에 대한 대학콤퓨터과학강의를 받고 UNIX에 기초한 C프로그람작성환경(실례로 cc, make, emacs 또는 vi 등등)에 대한 작업지식이 있어야 한다.

OSP는 매개 모듈이 장치일정작성, 처리기일정작성, 새치기조종, 파일관리, 기억기관리, 프로쎄스관리, 자원관리, 프로쎄스간통신과 같은 기본조작체계봉사를 수행하는 수

많은 모듈들로 구성된다. 강사는 임의의 부분적모듈들을 선택적으로 생략함으로써 대학생들이 잘못된 부분들을 완성하도록 하는 프로젝트들을 제기할수 있다. 이 과정은 OSP 프로젝트발생기에 의하여 완전히 자동화되며 배포판에 포함되여 있다. 프로젝트들은 강의내용과 모순이 없게 수행되도록 임의의 희망하는 순서로 조직할수 있다.

OSP프로젝트발생기는 강사에게 프로젝트들을 생성하기 위한 편리한 환경을 제공한다. 그것은 대학생들이 배당된 모듈들의 실현물을 련결하는 표준 OSP모듈들의 《부분적재모듈》을 발생한다. 결과 부분적으로는 대학생들이 완성한 새롭고 완전한 조작체계가 만들어 진다. 게다가 프로젝트발생기는 매개 배당된 모듈들을 위한 수속제목들과 필요한 자료구조들의 선언들을 포함하는《module.c》파일들을 생성한다. 이 파일들은 대학생들이 수속본체들에 작성해 넣는 프로젝트의 부분으로서 주어 질수 있다. 이것은 OSP와의 일관성 있는 대면부를 보증하며 강사와 대학생들이 타자하여야 할 루틴의 타자량을 작게 한다.

OSP의 심장부는 콤퓨터체계가 다중프로그람처리해야 할 동적으로 전개되는 사용자 프로쎄스들의 집합을 가지고 있는듯한 착각을 주는 모의기이다. OSP의 다른 모든 모듈들은 조작체계를 구동하는 모의기가 발생한 사건들에 적당히 응답하기 위하여 구축된다. 모의기는 흔히 자기가 모의한 사건에 대한 어떤 모듈의 응답오유를 검출할수 있는데 이러한 다른 모듈들과의 대화를 《리해한다》. 이러한 경우에 모의기는 사용자에게 중요한 오유통보문을 보내는것으로 프로그람의 집행을 친절하게 완료하고 오유가 일어 날수 있는 개소를 지적할것이다. 이 기능은 대학생들을 위한 착오수정도구로서 그리고 강사들을 위한 교육도구로서 봉사한다. 왜냐하면 이 기능이 모의기가 접수할수 있는 대학생프로그람이 실제상 착오가 없다는것을 보증하기때문이다.

모의기가 발생한 일감흐름들의 경쟁은 모의파라메터들을 조작하여 동적으로 조정될수 있다. 그것은 대학생프로그람들의 질을 검사하기 위한 단순하고 효률적인 방법을 준다. 또한 대학생들이 모의할 때 OSP와 대화하여 자기 프로그람들의 착오를 수정할수 있도록 하는 기능들이 있다.

OSP에서 기초를 이루는 모형은 임의의 특정조작체계의 복제가 아니다. 오히려 그것은 여러 체계들의 공통특징들을 추출한것이다(때때로 UNIX쪽으로 치우쳤다고 볼수 있지만). 프로쎄스간통신을 위한 모듈들을 제외한 OSP모듈들은 수많은 저준위사건들을 숨기기 위하여 설계되였으며 아직도 현대 체계들에서의 실제 대응모듈의 가장 명백한 견해들을 포함하고 있다. 그의 실현물들은 조작체계들에 대한 초보강의의 프로젝트부분품들로서 매우 적합하다. 프로쎄스간통신을 처리하는 두개 모듈을 중심으로 더 고급한 프로젝트를 구축할수 있다. 그것들의 설계는 더 세분화되여 대학생들에게 실제로《보다 불결한》 환경에서 연구할 기회를 준다.

OSP는 그의 호상작용이 그림 부4-1에서 설명된 10개의 1차모듈들로 구성된다. 이 그림에서 모듈사이의 화살표 즉 M에서 N으로의 화살표는 M의 루틴들이 N의 루틴들을 호출할수 있다는것을 의미한다. 다음의 목록에서 모듈들의 이름과 그의 간단한 기능설명을 제시한다.

SIMCORE	모의기의 핵심부
DIALOG	OSP와의 실행시대면부
INTER	일반새치기조종
IOSVC	입출력감시기호출
DEVINT	장치새치기
PAGEINT	페지부재새치기

PROCSVC 프로쎄스관리감시기호출
TIMEINT 시계새치기
MEMORY 기억기관리
CPU CPU일정작성
DEVICES 장치관리
FILES 파일조직
RESOURCES 자원관리
SOCKETS 프로쎄스간통신
PROTOCOLS SOCKETS모듈을 위한 규약지원

OSP에 대한 상세한 정보를 다음의 책에서 얻을수 있다.

OSP : An Environment for Operating System Projects, Michael kifer and Scott A. Smlka, Addison-Wesley, ISBN 0-201-54887-9(1991)

프로그람작성과제를 위한 제안들과 OSP의 설치 및 실행을 위한 방향들을 포함하고 있는 강사의 책판본(ISBN 0-201-54888-7)은 Addison-Wesley를 통해서만 강사들이 입수할수 있다.

강사들은 Addison-Wesley FTP싸이트 ftp://amstel.aw.com에서 강사지도서를 내리적재할수 있다. 가입 id와 통과암호는 Addison-Wesley판매대리인과 련계하여 얻을수 있다.

OSP는 SunOS, SystemV, BSD, 386BSD, Mach, Ultrix, HP-UX, AIX, Coherent 등과 같은 대부분의 워크스테이션 UNIX에서 실행한다. 그것은 또한 LINUX와 FreeBSD와 같은 PC UNIX에서도 실행한다.

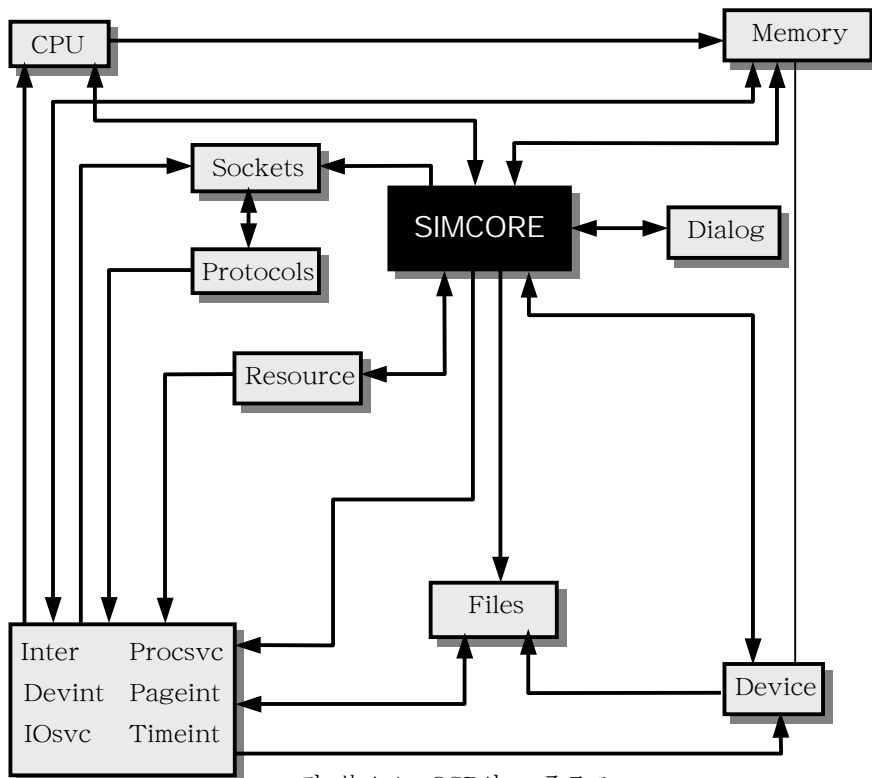



그림 부 4-1. OSP의 모듈구조

4-2. OSP의 혁신적인 측면

OSP의 주요한 혁신적인 측면은 다음과 같다. 즉

- OSP는 대학생들에게 현대 조작체계들의 많은 본질적인 특징들을 실현할수 있는 실험환경을 주는 동시에 그들을 저준위기계의존사건들과 분리시킨다. 따라서 대학생들은 한 학기동안에 얼마든지 가상기억기관리에서 페지교체전략들, 처리기일정작성전략들, 디스크탐색시간최적화 그리고 조작체계설계의 다른 문제점들을 배울수 있다. OSP의 심장부는 콤퓨터체계가 다중프로그람처리해야 할 동적으로 전개되는 사용자프로쎄스들의 집합을 가지고 있다는 착각을 주는 모의기이다. 대학생들이 학기강의에서 작성하는 OSP모듈들은 조작체계를 구동하는 모의기가 발생한 사건들에 적당히 응답하기 위하여 구축된다.

  대학생들은 OSP가 요청한 모의파라메터들에 서로 다른 값들을 주어 모의기가 발생한 일감흐름들의 경쟁을 조정할수 있다. 이것은 대학생들에게 자기 프로그람의 질을 검사하기 위한 간단하고 효률적인 방법을 제공한다. 또한 대학생들이 모의할 때 OSP와 대화하면서 자기 프로그람들의 착오를 수정할수 있게 하는 기능들이 있다.
- OSP는 실현물프로젝트들을 생성하기 위하여 편리한 환경인 OSP프로젝트발생기를 강사에게 제공한다. 프로젝트발생기는 대학생들이 배당된 모듈들의 실현물들에 련결하게 되는《부분적재모듈》과 같은 표준 OSP모듈들을 발생한다. 결과 대학생들이 부분적으로 완성한 새롭고 완전한 조작체계가 만들어 진다.

  게다가 프로젝트발생기는 매개 배당된 모듈들을 위한 수속표제들과 필수적인 자료구조들의 선언들을 포함하는《module.c》파일들을 자동적으로 생성한다. 이 파일들은 대학생들이 수속본체들에 작성해 넣는 프로젝트의 부분으로서 제공될수 있다. 이것은 OSP와의 일관성 있는 대면부를 보증하며 강사와 대학생들이 타자하여야 할 루틴의 량을 줄인다.
- OSP는 강사가 대학생들이 편리하고《안전한》방식으로 자기의 연구과제들을 제출하도록 하는데 사용할수 있는 hand-in프로그람을 제공한다. 주어 진 과제에서 hand-in프로그람은 대학생들이 OSP를 실현하는데 필요한 파라메터파일을 대학생들에게 요구할것이다. 이 파라메터파일(추측컨데 강사가 모순이 없는 기준에 기초하여 과제들의 등급을 정할수 있도록 성능검사한것)은 강사에 의하여 한번 생성될수 있거나 연구과제의 실현물에 대한 특정성능들을 설명하기 위하여 대학생들이 생성할수 있다. hand-in은 그다음에 대학생들의 원천프로그람들을 콤파일하고 주어 진 파라메터파일을 가지고 집행파일을 실행할것이다. 한번의 실행으로 완료했다면 hand-in은 사용자에게 또다른 파라메터파일을 요구할것이다. hand-in은 대학생들이 파일을 전혀 지정하지 않으면 탈퇴할것이다.

  대학생들이 완성한 원천모듈들과 번역단계와 모의실행들에서 나오는 출구는 강의보고에 있는 의뢰등록부에 넣어 질것이다. 대학생들은 이 출구파일들에 대한 그 어떤 호출권도 가질수 없으며 따라서 그것을 함부로 조작할수 없을것이다.

  대학생들은 자기 프로그람들을 여러번 제출할수 있다. 강사가 기한부를 확인할수 있도록 hand-in을 사용할 때마다 현재 시간을 기록한다. 그리고 매개 실행에서 hand-in은 강사가 준 파라메터파일로 수행되였는지 또는 대학생이 선택한 파라메터파일로 수행되였는지를 기록한다. 이것은 대학생들이 강사가 요구한 파라메터모임보다 《더 쉬운》 파라메터모임을 사용하지 못하도록 하는데 사용된다.

- OSP모의기는 대학생들이 실현한 모듈들의 실행시 행동을 주의깊게 감시하며 많은 경우에 대학생모듈의 행동이 표준행동에서 탈선한다면 경고통보문을 출구할수 있다. 실례로 대학생이 입출력요청들을 조종하는 모듈을 작성했다면 모의기는 입출력요청블로크(IORB)가 알맞는 장치대기렬에 삽입되였는가를 검사한다. 그렇지 않으면 모의기는 입출력요청이 정확히 조종되지 않았다는것을 대학생들에게 알리는 서술통보문을 출구한다.

  모의기가 수행하는 면밀한 감시기능은 대학생들이 대화식착오수정을 하는데 상당한 도움을 준다. 그것이 거의 항상 그 어떤 모의기경고통보문들을 발생시킴이 없이 대학생들의 OSP과제해답의 집행이 계속된다면 대학생들의 코드가 기능적으로 정확한 경우이다.
- OSP는 대학생들이 모의할 때 주기적으로 체계상태를 보게 하는(체계의 순시상기록을 취하도록 하는) 모의기와의 포괄적인 착오수정대면부를 제공한다. 속사기간에 연시된 정보에는 주기억기프레임표의 내용, 장치표, PCB(프로쎄스조종블로크)뿐, 사건대기렬 등이 포함된다. 순시상은 또한 사용자에게 모의파라메터들을 변화시키기 위한 기회를 준다. 이러한 파라메터의 하나인 Snapshot-intervay은 모의중에 체계상태를 어떻게 표시하겠는가를 지적하기 위하여 사용자가 설정할수 있다.
- OSP는 Addison-Wesley가 출판한 두개의 지도서인 OSP 참조안내서와 OSP 강사지도서에 충분히 서술되여 있다. 참조안내서는 대학생들에게 OSP연구과제를 완성하는데 필요한 모든 정보를 주며 매개 OSP모듈들에 대한 상세한 규격들과 연구과제들을 콤파일, 집행, 제출하기 위한 명령들을 포함한다. 강사지도서는 참고안내서에 보충적으로 SUNY Stony Brook에서 FTP를 통하여 OSP를 설치하기 위한 정보와 프로젝트발생기와 hand-in프로그람을 사용하는 방법에 대한 정보를 준다.

  게다가 강사지도서는 여러개의 표본 OSP프로그람작성과제를 수행하기 위한 실제본문(즉 설명과 지령들)을 포함한다. 그 실제본문은 지금까지 우리들과 다른 OSP사용자들이 사용하여 왔다. 다른 OSP사용자들이 제공한 과제들은 물론 이 과제들의 LaTex원천들은 배포판에 포함된다.
- 우리는 OSP사용자들의 전자우편목록을 가지고 있다. 우편목록의 목적은 OSP공동체에 쏘프트웨어와 문서의 모든 변경(실례로 착오수정, 개선된것, 예정판본)들을 통고하도록 하고 사용자들속에서 OSP를 론의하기 위한 매체로서 봉사하는것이다. 우편목록이 조작체계강의를 하는 강사를 위하여 마련된것이기때문에 임의의 우편목록신청자는 목록에 첨부되기전에 적당한 문서를 제출하여야 한다.

4-3. 다른 조작체계코스웨어와의 비교

조작체계강의안을 두개의 그룹 즉 모의기에 기초한것들(실례로 Berkeley의 소형조작체계와 Nochos, MPX)과 순수 기계에서 직접 실행하는 실제 OS의 원천코드에 기초한것들(실례로 MINIX[TANE97], XINU[COME84], LINUX)로 분류할수 있다. OSP는 명백히 첫 부류에 들어 간다. 두개 부류의 강의안을 조작체계교육의 두가지 다른 목적에 사용하는것을 볼수 있다. 순수 기계의 쏘프트웨어는 기계구성에 대한 저준위의 세부들과 밀접하게 친숙해 질수 있게 하며 모의기에 기초한 쏘프트웨어에서 존재할수 없는 직접성을 준다. 다른 한편 모의기에 기초한 쏘프트웨어는 대학생들을 특정기계구성의 진짜본질들과 고의적으로 분리시키며 대학생들이 강의 또는 강의안에서 론의하는 조작체계개념들을 실현하는데 집중하도록 한다.

모의기에 기초한 쏘프트웨어의 령역에서 OSP는 다음과 같은 성질들의 결합으로 하여 특징적이다.

- **유연성 :** 강사들은 제마음대로 자기가 마음에 드는 화제를 가지고 임의의 순서로 프로젝트들을 배당한다. 게다가 매개 프로젝트는 임의의 특정디스크 또는 처리기 일정작성, 기억기관리 또는 교착회피방책 등에 구속되지 않는다.
- **모의기에 의하여 제공되는 현실주의정도 :** OSP는 대표적인 조작체계에서 발생하는 사건들의 정확한 모의에 기초하며 이와 마찬가지로 강사는 대학생들이 실현한 과제들의 질을 평가할수 있도록 잘 준비된다. 모의기는 결과들을 위조하지 못하게 하며 따라서 대학생들의 프로젝트검출을 간단하게 하는 많은 보안검사기능들을 내장하고 있다.
- **사용의 용이성 :** 경험은 OSP가 강사와 대학생의 견지에서 상대적으로 사용하기 쉽다는것을 보여 주었다. 강사는 과제들을 작성하기 위한 관리적인 부담을 덜수 있다. 과제들은 프로젝트발생기가 자동적으로 발생하며 대학생들이 과제를 완성하는데 필요한 모든 정보를 포함한다. 모의기와의 착오수정대면부는 대학생들이 과제를 수행하는데 필요한 시간을 상당히 줄인다. 끝으로 hand-in프로그람은 강사들과 대학생들을 위하여 환경을 더 편리하게 한다.

4-4. OSP쏘프트웨어의 배포

OSP쏘프트웨어원천은 보고서 OSPftp인 ftp.cs.sunysb.edu에서 ftp를 통하여 입수할수 있다. 이 보고서는 OSP의 강사지도서에서 또는 Addison_Wesley판매대리인과 련계를 가지고 얻을수 있는 통과암호를 가진다(Addison_Wesley는 OS과목을 가르치는 강사들에게만 이 지도서를 제공한다.).

실제배포판은 파일

OSP. tar. Z

에 있다. 이것은 압축된 tar파일이다. 원천을 설치하기 위하여 압축해제와 tarxf를 사용하시오. 그다음에 쉘 script bin/osp.startup와 bin/osp.compile을 집행하여 OSP를 콤파일할수 있다. 더 상세한것은 OSP강사지도서를 보시오.

OSP는 전용보고서의 뿌리등록부에서 가장 잘 실행한다. 그러나 HOME을 알맞게 설정하면 보조등록부에서도 실행할수 있다.

배포판의 등록부구조는 OSP$모듈들의 원천코드(역시 표본해답으로서도 봉사한다.) OSP.startup와 OSP.compile과 같은 여러가지 쉘스크립트들, 수많은 제안된 OSP프로그람작성과제들 등으로 구성된다.

4-5. OSP의 우편목록

OSP사용자들을 위하여 우편목록이 존재한다. 우편목록의 목적은 쏘프트웨어와 문서에서의 변화들(실례로 착오상태, 예정판본)을 OSP공동체에 통지하도록 하고 OSP사용자들속에서 진행되는 OSP에 대한 론의점들을 위한 매체로서 봉사하는것이다.

이 우편목록은 조절될수 있으며 모든 통신들은 OSP@cs.Sunysb.edu에 송신될수 있다. 이 주소는 또한 임의의 문제점들을 보고하고 일반적으로 관심사로 되지 않는 OSP관련문제들을 질문하는데 그리고 우편목록관리문제(실례로 가입의 추가/삭제)를 위하여

사용될수 있다.

우편목록은 다만 강사가 OSP쏘프트웨어를 사용한 조작체계강의에 사용하기 위한것이라는것을 명심하시오. 특히 조작체계강의를 받는 대학생들을 위한것은 아니다. 그러므로 독자들은 우편목록을 구독하고 싶다면 OSP@cs.Sunysb.edu에 이름, 위치, 주소, 전화번호를 전자우편으로 보내고 동시에 부서소책자사본 또는 이름이 적힌 강의일정을 우편으로 보내시오.

4-6. 전망계획

현재는 림시적으로 OSP/2라고 하는 다음세대 OSP쏘프트웨어를 설계 및 실현하고 있다. 이 새로운 판본은 대학생들에게 현대적인 조작체계설계 및 실현물을 위한 보다 현실적이고 현대적인 관점을 주고 체계의 모듈성과 확장성을 증대시키며 객체지향설계를 받아 들이는 등 여러가지 측면에서 OSP보다 우수하다. OSP/2는 JAVA로 작성하고 있으며 대학생들은 OSP/2프로젝트를 JAVA로 프로그람작성할수 있다.

OSP/2의 예정배포날자는 2001년 9월이다

부록 5. BACI : Ben-Ari 병행프로그람작성체계

5-1. 서론

제5장에서는 병행개념(호상배제와 림계구역문제)을 소개하고 있으며 동기화수법(신호기들, 감시기들과 통보문넘기기)을 제기하고 있다. 병행프로그람의 교착 및 고갈문제들은 제6장에서 취급하고 있다. 병렬 및 분산계산에 대한 중요성이 커짐에 따라 병행 및 동기화문제를 리해할 필요성이 더욱 제기되고 있다. 이러한 개념을 충분히 리해하기 위해서는 병행프로그람을 실지 작성해 본 경험을 가지고 있어야 한다.

이《실제적인》경험을 얻기 위한 세가지 방법이 있다. 첫째로, 병행 Pascal, Modula, Ada 또는 SR프로그람작성언어와 같은 병행프로그람작성언어로 병행프로그람들을 작성해 보는것이다. 그러나 각이한 동기화수법들을 실험하기 위해서는 많은 병행프로그람작성언어들의 문장구성법을 배워야 한다. 둘째로, UNIX와 같은 조작체계의 체계호출들을 사용하여 병행프로그람을 작성해 보는것이다. 그러나 특정한 조작체계의 세부들과 특성들(UNIX의 신호기체계호출들의 세부들)로 하여 병행프로그람작성법을 리해하려는 목적으로부터 벗어나기가 쉽다. 마지막으로 기능병행 Pascal(Pascal-FC) [DAVI90]이나 Ben-Ari병행해석기(BACI)[BYNU96]과 같은 병행개념을 체험시키기 위하여 개발된 언어로 병행프로그람들을 작성해 보는것이다. 여기서는 그러한 언어를 사용하여 일상생활에서 친숙해 진 문장론에 의한 각이한 동기화수법들을 제공하고 있다. 병행개념들에 대한 지식을 주기 위하여 전용으로 개발된 언어들은 필요한 지식을 습득하기 위한 가장 좋은 수단으로 된다.

부록 5-2에서는 BACI체계에 대하여 간단히 개괄하고 체계를 습득하기 위한 방법을 설명한다. 부록 5-3에서는 BACI프로그람실례들을 제시하고 부록 5-4에서는 실현 및 프로그람작성준위에서 실제적인 병행지식을 얻기 위한 프로젝트를 취급한다. 마지막으로 부록 5-5에서는 진척중에 있거나 계획되여 있는 BACI체계의 갱신에 대하여 서술한다.

5-2. BACI

체계에 대한 개괄

BACI는 순차 Pascal(Pascal-S)에 Ben-Ari가 변경을 가하여 얻은 그의 직접적 후손이다. Pascal-S는 Wirth에 의하여 작성된 표준 Pascal의 부분집합으로서 INPUT와 OUTPUT, 묶음들, 지시자변수들, goto문장들을 제외하고는 파일들이 없다. Ben-Ari는 Pascal-S언어에 **cobegin... cend**구문과 같은 병행프로그람작성구문들과 그리고 **기다림** 및 **신호**조작들이 달린 신호기변수형태를 보충하였다[BEN82]. BACI는 Pascal-S에 동기화기능들(실례로 감시기들)과 그리고 사용자가 변수를 부적당하게 변경시키지 못하도록 하기 위하여 은페기구들(실례로 신호기변수는 신호기함수들에 의해서만 변경시킬수 있다)을 보충한 Ben-Ari의 변경판이다.

BACI는 병행프로쎄스집행을 모의하며 일반신호기들, 감시기들과 같은 동기화수법들을 제공하고 있다. BACI체계는 그림 부 5-1에서 보는바와 같이 두개의 부분체계들로 구성되여 있다. 첫번째 부분체계인 콤파일러는 사용자의 프로그람을 PCODE라고 부르는 중간적인 목적코드로 콤파일한다. BACI체계에서는 두개의 콤파일러를 사용할수 있는데 이것은 기초프로그람작성강의들에서 가르친 두개의 통속적인 언어에 대응한다. 한 콤파일러의 문법은 표준 Pascal과 류사한데 Pascal문법을 사용하는 BACI프로그람을 pgrm-name.pm으로 표기한다. 다른 콤파일러의 문법은 표준 C++와 류사한데 이 BACI 프로그람들을 pgrm-name.cm으로 표기한다. 콤파일러들은 둘다 콤파일기간에 두개의 파일 즉 pgrm-name.lst와 pgrm-name.pco를 창조한다.

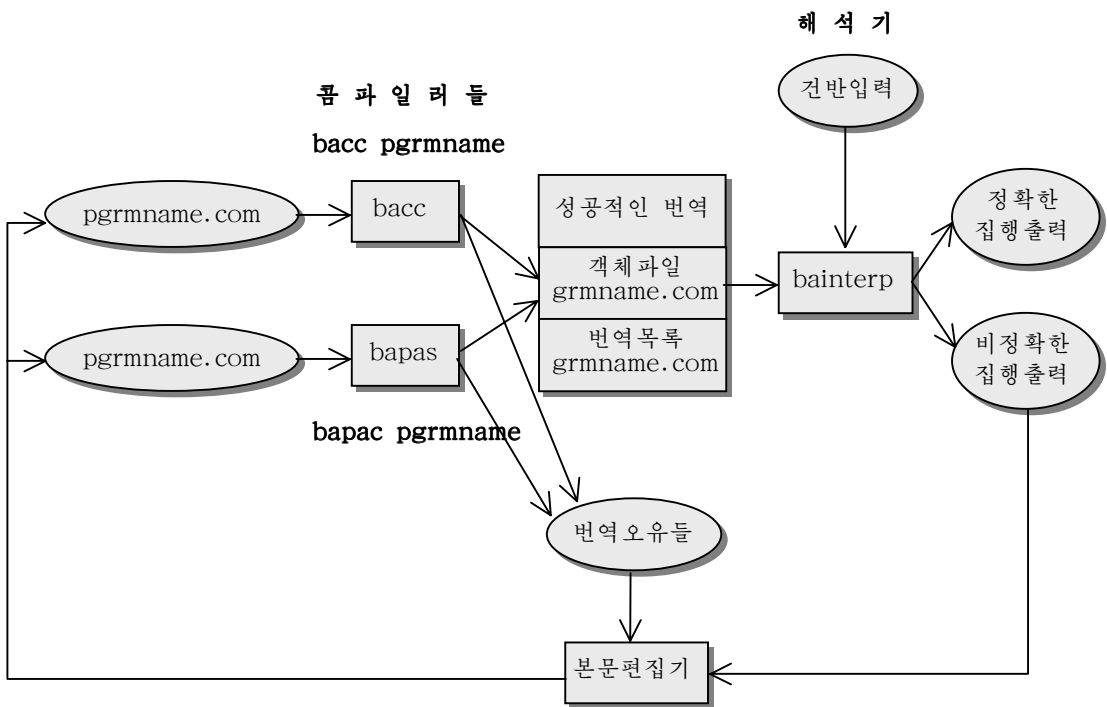



그림 부 5-1. BACI 체계의 개괄

BACI체계에서 두번째 부분체계인 해석기는 콤파일러가 창조한 목적코드를 집행한다. 즉 해석기는 pgrm-name.pco를 집행한다. 해석기의 핵심은 선취일정작성기로서 집행기간에 이 일정작성기는 병행프로쎄스들사이에서 임의로 교체하면서 병행프로쎄스들에 대한 병렬집행을 모의한다. 해석기는 단일걸음집행, PCODE명령의 역아쎔블프로그람의 기억위치표시와 같은 많은 각이한 착오수정방법들을 제공한다.

BACI에서 병행구문

이 목록의 나머지 부분에서는 표준 C++와 류사한 콤파일러에 중심을 둔다. 이 콤파일러를 C--고 부르는데 그 문법은 C++와 류사하기는 하지만 계승성, 은폐 및 다른 객체지향프로그람작성의 특징들은 포함하지 않는다. 여기서는 BACI의 병행구문에 대하여 개괄하며 Pascal이나 C--ACI문법에 대하여 더 구체적인것이 필요하다면 BACI 웨브싸이트에서 사용자의 안내를 보시오.

Cobegin

병행으로 실행되는 프로쎄스의 목록은 Cobegin블로크안에 있다. 이러한 블로크들은 중첩될수 없으며 주프로그람에 있어야 한다.

Cobegin { proc 1(...); proc 2 (...); ... ; proc N(...); }

우의 블로크에서 콤파일러가 창조한 PCODE문은 해석기에 의해 독단적으로,《임의》의 순서로 교차처리되는데 Cobegin블로크를 포함하는 같은 프로그람의 다중집행은 비결정론적인것으로 될것이다.

신호기

BACI에서 신호기는 부아닌 값을 가지는 옹근수변수로서 뒤에는 정의되여 있는 신호기의 호출에 의해서만 접근할수 있다. BACI에서 2진신호기는 단지 0과 1값만 가진다고 가정한것으로서 신호기형중에서 2진신호기의 부분형에 의하여 지원을 받는다. 콤파일 및 집행기간에 콤파일러와 해석기는 2진신호기의 변수가 0과 1값만을 가질수 있다는 제한조건 또는 신호기의 형이 다만 부가 아닐수 있다는 제한조건을 실시한다. BACI신호기 호출에는 다음과 같은것들이 속한다.

- Initialsem(신호기 sem, int표현)
- P(신호기 sem) : sem의 값이 0보다 크면 해석기는 sem을 하나 감소시키고 P의 호출자가 계속할수 있게 한다. Sem의 값이 0과 같다면 해석기는 P의 호출자를 잠자게 한다. Wait는 P에 대한 동의어로 볼수 있다.
- V(신호기 sem) : sem의 값이 0과 같고 하나 혹은 그이상의 프로쎄스들이 sem에 관하여 잠자고 있다면 이 프로쎄스들중 하나를 깨운다. 아무런 프로쎄스도 sem에 관하여 기다리고 있지 않다면 sem을 하나 증가시킨다. 임의의 사건에서 V의 호출자는 계속하도록 허락을 받는다(BACI는 어떤 신호가 도착할 때 어느 프로쎄스를 깨우겠는가 하는것을 우연적으로 선택하여 Dijkstra의 원래의 신호기제안에 따른다.). 신호를 V에 대한 동의어로 볼수 있다.

감시기

BACI는 Hoare[Hoar74]에 의해 제안된바와 같이 감시기의 개념을 일정한 제한조건을 가지고 지원하는데 그 실현물은 [PRAM84]에 의해 수행된 작업에 기초하고 있다. 감시기는 수속이나 함수에 의하여 정의된 블로크와 같이 일부 추가적인 속성(실례로 조

건변수들을 들수 있다.)들을 가지는 C++의 블로크이다. BACI에서 감시기는 제일 바깥인 전역준위에서 선언되여야 하며 다른 감시기의 블로크와 중복될수 없다. 세개의 구문이 병행성을 조종하기 위해 감시기의 수속들 및 기능들에 의하여 사용되고 있다. 즉 조건변수들, waitc(어떤 조건을 기다린다.) 및 signalc(어떤 조건을 신호한다.)들에 의하여 사용된다. 조건은 사실상 어떤 값을 절대로《가지지》않으며 대기할곳이 어디이며 신호할것이 어떤것인가 하는것이다. 감시기프로쎄스는 waitc와 signalc호출들을 통하여 주어진 조건이 유효하기를 기다리거나 주어 진 조건이 이제부터 유지된다는것을 신호할수 있다. waitc와 signalc의 호출들은 다음의 문맥과 의미론을 가지고 있다.

- waitc(조건cond, int prio) : 감시기프로쎄스(그리고 감시기프로쎄스를 호출하는 외부프로쎄스)는 조건 cond에 의하여 폐색되며 우선권 prio를 할당 받는다.
- waitc(조건 cond) : 이 호출은 우에서 말한 waitc호출과 같은 의미를 가지지만 기정우선권을 10으로 할당 받는다.
- signalc(조건 cond) : 가장 작은 (가장 높은) 우선권을 가지고 cond에 관하여 기다리고 있는 어떤 프로쎄스를 깨운다. cond를 기다리고 있는 프로쎄스가 전혀 없으면 아무것도 하지 않는다.

BACI는 즉시회복요구에 따른다. 즉 어떤 조건에 의하여 기다리고 있는 프로쎄스가 신호를 받았다면 감시기에 들어 가려고 하는 프로쎄스보다 높은 우선권을 가진다.

다른 병행성구문

C-BACI콤파일러는 새로운 병행조종기본지령들을 생성하는데 사용할수 있는 몇가지 낮은 준위의 병행구문을 제공한다. 기능이 원자적으로 정의된다면 그 기능은 비선취적이다. 다른 말로 해석기가 문맥절환을 사용하여 원자기능을 새치기하지 못한다. BACI에서 중단기능은 호출하는 프로쎄스를 잠자게 하고 회복기능은 중단된 프로쎄스를 회복시킨다.

BACI를 얻는 방법

두개의 사용자안내서(두개의 콤파일러에 각각 하나)와 상세화된 프로젝트서술을 가진 BACI체계는 BACI Web싸이트(William Stallings.com/OS4e.html에서 그것들의 Web싸이트에 련결할수 있다.)에서 입수할수 있다. BACI체계는 C로 작성되여 있으며 따라서 쉽게 이식할수 있다. 현재 BACI체계는 Makefile파일의 약간한 변경으로 LINUX, AIX및 DOS에서 콤파일할수 있다(주어 진 가동환경에 대한 구체적인 설치자료는 배포판의 README를 보시오.). bynum@cs.wm.edu 혹은 tcamp@mines.edu에서 자료를 얻을수 있다.

5-3. BACI프로그람의 실례

제5장과 제6장에서 많은 고전적인 동기화문제들을 설명하였다(실례로 읽기자/쓰기자문제와 철학자식사문제를 들수 있다.). 여기서는 세가지 BACI프로그람을 사용하여 BACI체계를 설명한다. 첫 실례는 BACI체계에서 병행프로쎄스의 집행에서의 비결정론을 설명한다. 다음의 프로그람을 고찰하자.

```
const int m = 5;
int n;
void incr(char id)
```

```
{
    int i;
    for (i =1; i < = m; i = i+1)
    {
            n = n+1;
            cout << id << "n =" << n << " i = ";
            cout << I << "  "<< id <<endl;
    }
}
main(  )
{
    n=0;
    cobegin{
            incr( 'A'); incr('B'); incr('C');
    }
    cout << 《합은 다음과 같다.》 < n < endl;
}
```

앞의 프로그람에서 창조된 세개의 프로쎄스(A, B, C)가 각각 차례로 집행되였다면 출력합은 15로 된다는것을 주목하자. 그러나 명령문 n=n+1의 병행집행은 서로 다른 값을 가지는 출력합을 초래할수 있다. bacc를 사용하여 앞의 프로그람을 콤파일한후에 bainterp으로 PCODE파일을 여러번 집행하였다. 매 집행은 9～15사이의 출력합을 산생시켰다. BACI해석기가 산생시킨 하나의 실례집행은 다음과 같다.

```
원천파일 : incremen.cm Fri Aug 1 16 : 51 : 00 1997
CB  n = 2  i = 1  C  n = 2
A  n = 2  i = 1  i = 1 A
CB
     n = 3  i =2  C
A  n = 4  i = 2  C  n = 5  i = 3  C
A
B  n =6C  i = 2  B
     n = 7  i = 4  C
A  n = 8  i = 3  A
BC  n = 10  n = 10  i = 5  C
A  n = i = 311  i = 4  A
 B
A  n = 12  i = B5  n = 13A
   i = 4  B
B  n = 14  i = 5  B
합은 14이다.
```

공통 주기억기에 대한 프로쎄스의 접근을 동기화시키자면 특수한 기계명령이 필요하다. 그다음 이 특수한 기계명령우에 호상배제규약들 또는 동기화기본지령을 구축한다. BACI에서 해석기는 원자적으로 정의된 기능을 문맥절환으로 새치기할수 없다. 이 특징으로 하여 사용자는 낮은 준위의 특수 기계명령을 실현할수 있다. 실례로 다음의 프로그람이 제5장 제3절에서 정의된 기능에 대한 BACI실현물이다.

```
//검사 및 설정명령
//Stallings, 제 5장 제3절
//
atomic int testset (int & i)
{
      if (i = =0) {
       i = 1;
       return 1;
      }
      else
       return 0;
}
```

우리는 testset를 사용하여 다음의 프로그람에서 보여 준바와 같이 호상배제규약을 실현할수 있다. 이 프로그람은 검사 및 설정명령에 기초한 호상배제프로그람의 BACI실현물이다. 프로그람은 세개의 병행프로세스를 가정하며 매개 프로세스는 호상배제를 10번 요구한다.

```
int bolt = 0;
const int RepeatCount = 10;
viod proc(int id)
{
      int i = 0;
      while ( i < RepeatCount) {
        while (testset (bolt)); //대기
              //림계구역에 들어 간다
              cout << id;
      //림계구역을 벗어 난다
      bolt = 0;
              i++;
      }
}

main (  )
{
      cobegin{
        proc(0); proc(1); proc(2);
      }
}
```

다음의 두개 프로그람은 신호기에 의한 경계완충기식생산자/소비자문제의 BACI풀이이다(그림 부 5-17을 보시오.). 이 실례에서는 두개의 생산자, 세개의 소비자가 있으며 완충기의 크기는 5이다. 우선 이 문제에 대한 프로그람의 세부들을 서술한다. 다음 경계완충기식실현물을 정의하는 포함파일을 서술한다.

```
//경계완충기식생산자/소비자문제에 대한 풀이
```

```
//stallings 5-17
//경계완충기식기계에 가져다 넣는다
#include "boundedbuff.inc"
const int ValuRange = 20; // 0∼19사이에 있는 옹근수들이 산생된다.
semaphore to; //말단출력에로의 배타접근을 위하여
semaphore s; //완충기에 대한 호상배제
semaphore n; // #완충기에서 소비할수 있는 항목들
semaphore e; // #완충기에서 빈공간
int produce( char id)
{
        int tmp;
        tmp = random(ValueRange);
        wait(to);
        cout<<: "생산자" << id << "생산한다" << i << endl;
        signal(to);
        return tmp;
    }
    void consume( char id, int i)
    {
        wait (to);
        cout << "생산자" << id << "소비한다" << i << endl;
        signal(to):
    }
    void producer (char id)
    {
        int i;
        for (; ;) {
            i =  produce(id);
            wait(e);
            wait(s);
            append(i);
            signal(s);
            signal(n);
        }
    }
    void consumer(char id)
    {
        int i;
        for (;;) {
            wait(e);
            wait(s);
            I=take( );
            signal(s);
            signal(n);
            consume (id, i);
        }
```

```
}
main( )
{
    initialsem (s, 1);
    initialsem (n, 0);
    initialsem (e, SizeOfBuffer);
    initialsem (to, 1);
    cobegin {
       producer('A'); producer('B');
       consumer('x'); consumer('y'); consumer('z');
    }
}
//경계완충기파일포함
const int SizeOfBuffer = 5;
int buffer[SizeOfBuffer];
int in = 0; //다음번 append에 사용되는 완충기에 대한 첨수
int out = 0; //다음번 take에 사용되는 완충기에 대한 첨수
void append ( int v)
   // 완충기에 V를 더한다
   // 신호기나 조건을 통하여 외적으로 초과량을 관심하는것으로 가정한다.
{
    buffer[in] = v;
    in = (in + 1) % SizeOfBuffer;
}
int take( )
   //완충기에서 항목을 반환한다
   //신호기나 조건을 통하여 외적으로 부족량을 관심하는것으로 가정한다
{
    int tmp;
    tmp = buffer[out];
    out = (out + 1) % SizeOfBuffer;
    return tmp;
}
```

BACI에서 앞의 경계완충기식풀이에 대한 하나의 실례집행은 다음과 같다.

```
Source file: semprodcons.cm Fri Aug 1  12 : 36 : 55  1997
Producer B produces 4
Producer A produces 13
Producer B produces 12
Producer A produces 4
Producer B produces 17
Consumer x consumes 4
Consumer y consumes 13
Producer A produces 16
Producer B produces 11
```

Consumer z consumes 12
Consumer x consumes 4
Consumer y consumes 17
Producer B produces 6
…

5-4. BACI프로젝트

여기서는 BACI에서 실현할수 있는 두가지 일반형프로젝트들을 설명한다. 우선 낮은 준위조작에 대한 실현물을 포함하는 프로젝트들을 설명한다(실례로 어떤 공통 주기억기에 대한 프로쎄스의 접근을 동기화시키는데 쓰이는 특수한 기계명령을 들수 있다.). 그 다음 이 낮은 준위조작들의 우에 구축되는 프로젝트들을 설명한다(실례로 고전적인 동기화문제들을 들수 있다.). 이 프로젝트들에 대한 보다 구체적인 내용은 [BYNU96]과 BACI배포판에 포함된 프로젝트서술내용을 보시오. 이 프로젝트들중 몇개를 해결하기 위해 교원들은 저자들과 련계를 가져야 할것이다. 이 절에서 설명된 프로젝트들외에 제5장의 마감에 있는 많은 문제들을 BACI로 실현할수 있다.

동기화기본지령의 실현

기계명령의 실현

BACI로 실현할수 있는 많은 기계명령들이 있다. 실례로 그림 부5-5에서 준 교환명령이나 [HERL90]에서 준 비교 및 교체조작을 실현할수 있다. 이 명령들의 실현은 int값을 되넘기는 원자기능에 기초하고 있다. 낮은 준위조작의 우에 호상배제규약을 구성하여 기계명령의 실현물들을 검사할수 있다.

공평한 신호기의 실현(FIFO)

BACI에서 신호기조작은 임의의 깨우기순서를 사용하여 실현되는데 이것은 Dijkstra가 신호기를 처음으로 정의한 방법이다. 그러나 제5장 제4절에서 설명한것처럼 가장 공평한 방책은 FIFO이다. 이 FIFO깨우기순서를 사용하여 BACI에서 신호기들을 실현할수 있다. 그 실현에서는 적어도 다음의 네가지 수속을 정의해야 한다.

- 프로그람코드를 초기화시키기 위한 CreateSemaphores( )
- sem-index로 표현된 신호기를 초기화하기 위한 InitSemaphore(int sem -index)
- FIFOP(int sem-index)
- FIFOV (int sem-index)

이 코드는 체계실현으로서 작성되여야 하며 따라서 가능한 모든 오유들을 처리해야 한다. 즉 신호기설계자는 사용자공동체에 의한 무식하고, 어리석은 또는 부당한 사용을 이겨낼수 있는 코드를 산생하여야 한다.

신호기, 감시기 및 실현물들

많은 고전적인 병행프로그람처리문제 즉 생산자/소비자문제, 각이한 우선권을 가지 는 철학자문제, 읽기자/쓰기자문제, 잠자는 리발사문제 및 흡연자문제들이 있다. 이 모든 문제들을 BACI에서 실현할수 있다. 여기서는 병행성과 동기화개념에 대한 리해를 더 깊이 하기 위하여 BACI로 실현할수 있는 비표준신호기/감시기프로젝트들을 설명한다.

A와 B의 신호기들

BACI에 있는 다음의 프로그람륜곽에 대하여

```
//여기서 전역신호기선언
void A( )
{
   p( )'s and v( )'s ONLY
}
void B( )
{
   p(  )'s 및 v(  )'s ONLY
}
main( )
{
  //여기서 신호기를 초기화한다.
  cobegin{
     A( ); A( ); A( ); B( );B( );
  }
}
```

은 프로쎄스들이 A(어떤 복사), B(어떤 복사), A(어떤 복사), A, B의 순서로 결속하는 식으로 가장 적은 수의 일반신호기들을 사용하여 프로그람을 완료한다. 해석기의 선택위치 –t를 사용하여 프로쎄스의 완료를 현시하시오(이 프로젝트에 대한 많은 변종들이 있다. 실례로 ABAA순서로 완료하는 네개의 병행프로쎄스들을 가지거나 AABABABB순서로 완료하는 여덟개의 병행프로쎄스들을 가지는 프로젝트들이 있다.).

2진신호기의 사용법

2진신호기를 사용하는 앞의 프로젝트를 다시 고찰하시오. 할당 및 IF-THEN-ELSE문이 이전 프로젝트들에 대한 풀이들에서 필요없다고 하여도 이 풀이에서 필요한 리유를 평가해 보시오. 즉 이 경우에 Ps 및 Vs만을 사용할수 없는 리유를 설명하시오.

차지기다림 대 신호기

신호기들을 사용하는 풀이와 차지기다림(실례로 testset명령)를 사용하는 호상배제에 대한 풀이의 성능을 비교하시오. 실례로 이전에 설명한 ABAAB프로젝트에 대한 신호기 풀이와 testset풀이를 비교하시오. 매 경우에 집행수를 많이 택하여(가령 1000) 더 좋은 통계들을 얻어 내시오. 결과를 론의하고 하나의 실현물이 다른 실현물보다 우월한 리유를 설명하시오.

신호기와 감시기

제5장의 문제 23의 자료에 기초하여 일반신호기를 사용하는 감시기를 실현한 다음 BACI에서 감시기를 사용하는 일반신호기를 실현하시오.

일반신호기와 2진신호기

한 형태의 신호기를 사용하여 다른 형태의 신호기를 실현하거나 혹은 그 반대로 실현하여 일반신호기와 2진신호기가 다 같이 위력하다는것을 증명하시오.

시간박자 : 감시기의 프로젝트

[SIBL98]의 제6장 문제 16과 류사하게 감시기 AlarmClock을 포함하는 프로그람을 작성하시오. 감시기는 int형 변수 the Clock(령으로 초기화됨)와 다음의 두가지 함수을 가져야 한다.

- Tick( ) : 이 함수는 호출될 때마다 the Clock를 증가시킨다. 이것은 필요하다면 signalc와 같이 다른 일들을 수행할수 있다.
- Int Alarm(int id, int delta) : 이 함수는 the Clock의 최소한의 델타박자동안에 식별자 id를 사용하여 호출자를 폐색시킨다.

주프로그람은 다음의 두가지 함수도 가질수 있다.

- Void Ticker( ) : 이 수속은 영구적인 반복고리에서 Tick( )를 호출한다.
- Void Thread(int id, int myDelta) : 이 함수는 영구적인 반복고리에서 Alarm을 호출한다.

독자들은 감시기가 요구하는 임의의 다른 변수들을 감시기에 부여할수 있다. 감시기는 다섯개까지의 동시적인 경보를 받아 들일수 있어야 한다.

대중빵집문제

최근에 빵굽기가 류행되는것으로 하여 거의 모든 손님들은 봉사를 받기 위하여 기다리려고 한다. 봉사를 계속하기 위해 빵집은 손님들이 차례로 봉사 받을수 있게 해줄 표체계를 확립하려고 한다. 이 표체계에 대한 BACI실현물을 연구하시오.

5-5. BACI체계의 확대

BACI체계를 확장하는데는 다음과 같은 몇가지 방도가 있다.

1. BACI의 UNIX판본에 도형사용자대면부(GUI)를 추가하였다. 이 GUI로 하여 사용자는 같은 체계에서 BACI프로그람모두를 편집, 콤파일, 해석할수 있다. 천연색창문들은 BACI프로그람의 집행을 설명해 준다. BACI GUI는 http://www.mines.edu/fs_home/tcamp/GUI/index.html에서 입수할수 있다.
2. BACI의 분산판본을 생성하였다. 병행프로그람과 마찬가지로 분산프로그람을 실현함이 없이 분산프로그람들의 정확성을 증명하기는 어렵다. 분산 BACI는 분산프로그람들을 쉽게 실현할수 있도록 한다. 분산프로그람의 정확성을 증명하는외에 분산 BACI를 사용하여 프로그람의 성능을 검사할수 있다. 분산 BACI는 http://www.mines.edu/fs_home/tcamp/dbaci/index.html 에서 입수할수 있다.
3. 우리는 사용자에게 PCODE파일에 대한 주석이 달린 목록을 제공하며 매개 PCODE명령의 기계코드와 필요하다면 명령을 발생한 대응프로그람원천을 보여주는 PCODE역아쎔불러를 가지고 있다.
4. 두 콤파일러에 독자적인 콤파일능력과 외부변수들을 추가하였다. 현재 BACI PCODE의 서고를 창조하고 사용할수 있게 할 련결편집기와 파일보관프로그람을 개발하고 있다.

용 어 해 설

가상기억기　Virtual Stroage

가상주소들이 실제주소들로 사영되는 콤퓨터체계의 사용자에 의하여 주소지정가능한 주기억기로서 간주될수 있는 기억기공간. 가상기억기의 크기는 주기억기세포들의 실제적인 개수에 의해서가 아니라 콤퓨터체계의 주소지정방식과 사용가능한 2차기억기의 크기에 의하여 제한된다.

가상주소　Virtual Address

가상기억기에서 기억위치의 주소

감시기　Monitor

추상적인 자료형태들과 수속들들과 모임에 대한 호상배제적인 접근을 제공하는 프로그람작성언어구문

강한 신호기　Strong Semaphore

같은 신호기우에서 기다리고 있는 모든 프로쎄스들이 기다림(P)조작(FIFO순서)들을 집행할 때와 같은 순서로 대기렬을 짓고 있다가 드디여 진행되도록 하는 신호기

경량프로쎄스　Lightweight Process

스레드

경쟁조건　Race Condition

다중프로쎄스들이 프로쎄스들의 상대적인 동기에 의거하는 결과에 따라 공유자료를 호출하고 조작하는 상황

고갈　Starvation

다른 프로쎄스들에 항상 선취권이 주어 지기때문에 어떤 프로쎄스가 무한정 지연되는 상태

교착　Deadlock

여러개의 프로쎄스들이 류사한 기다림상태에 있는 다른 프로쎄스에 의하여 유지되고 있음으로 하여 사용할수 없게 되여 있는 자원을 사용하기 위해 기다리고 있는 경우에 발생하는 곤경(1). 여러개의 프로쎄스들이 류사한 상태에 있는 다른 프로쎄스의 작용이나 그것의 응답을 기다리는 경우에 발생하는 곤경(2)

교착검출　Deadlock Detection

요청된 자원들이 사용가능할 때 언제든지 허용하는 수법. 주기적으로 조작체계는 교착을 검사한다.

교착예방　Deadlock Prevention

교착이 발생하지 않도록 담보하는 수법. 예방은 교착의 필요한 조건들중의 한가지를 만족시키지 않도록 하는 방법으로 실현된다.

교착회피　Deadlock Avoidance

교착에 대한 매개 새로운 자원요청을 조사하는 동적수법. 만일 새로운 요청이 교착에로 이끌어 간다면 그러한 요청은 거절된다.

교체 Swapping
주기억기령역의 내용들을 2차기억기령역에 있는 내용으로 서로 바꾸는 프로쎄스

규약자료단위 Protocol Data Unit
망의 동위실체들사이에서 전달되는 단위로서 조종정보. 주소정보 또는 자료를 포함할 수 있다.

금지형새치기 Disabled Interrupt
처리기가 특정한 부류의 새치기요청신호들을 무시하는 조건으로서 보통 조작체계에 의하여 창조된다.

기억기분할 Memory Partitioning
기억기를 독립적인 구간들로 세분하는것

기준주소 Base Address
콤퓨터프로그람의 집행에서 주소계산의 기준으로 사용되는 주소

개방체계호상련결(OSI)참조모형 Open Systems Interconnection (OSI) Reference Model
협동하는 장치들사이의 통신모형. 그것은 7층짜리 통신기능방식을 정의한다.

객체요청중개자 Object Request Broker
의뢰기에서 봉사기에로 송신된 요청들에 대한 중개자로서 작용하는 객체지향체계의 실체

과도교체 Thrashing
처리기가 명령들의 집행보다 토막들의 교체에 대부분의 시간을 소비하는 가상기억기기구에서의 현상

과제 Task
프로쎄스와 동일

내부쪼각 Internal Fragmentation
기억기를 고정된 크기의 분할구역(즉 주기억기의 페지프레임이나 디스크의 물리적인 블로크)들로 나눌 때 발생한다. 만일 자료블로크가 한개이상의 분할구역들에 할당된다면 마지막분할구역의 공간을 랑비할수 있다. 이것은 자료의 마지막부분이 마지막 분할구역보다 작을 때 발생한다.

다중과제처리 Multitasking*
두개 또는 그이상의 콤퓨터과제들을 병행수행 또는 교차집행하기 위하여 제공하는 조작방식

다중준위보안 Multilevel Security
다중준위의 자료분류를 거쳐 접근조종하게 하는 능력

다중처리 Multiprocessing*
다중처리기에서 두개 또는 그이상의 처리기들에 의하여 병렬처리를 하기 위하여 제공하는 조작방식

다중처리기 Multiprocessor
주기억기에 공동으로 접근하는 두개 또는 그이상의 처리기를 가진 콤퓨터

다중프로그람처리 Multiprogramming

단일처리기로 두개 또는 그이상의 콤퓨터프로그람들을 교대로 집행하기 위하여 제공하는 조작방식

다중프로그람처리준위 Multiprogramming Level

주기억기에 부분적으로 또는 완전히 상주하는 프로쎄스의 개수

단일핵심부 Monolithic Kernel

일정작성, 파일체계, 장치구동프로그람. 기억기관리를 비롯한 실제상 완전한 조작체계를 포함하는 큰 핵심부. 핵심부의 모든 기능요소들은 자기의 모든 내부자료구조들과 루틴들에 접근한다. 대표적으로 단일핵심부는 같은 주소공간을 공유하는 모든 요소들을 가진 단일프로쎄스로써 실현된다.

동기조작 Synchronous Operation

다른 프로쎄스에서의 규정된 사건의 발생에 관하여 규칙적으로 또는 예상할수 있게 발생하는 조작. 규정된 사건의 실례로는 콤퓨터프로그람의 미리 코드화된 위치에서 조종을 넘겨 받는 입출력루틴의 호출을 들수 있다.

동기화 Synchronization

두개 또는 그이상의 프로쎄스들이 어떤 조건에 따라 자기들의 동작들을 일치시키는것

동적재배치 Dynamic Relocation

프로그람이 주기억기의 각이한 령역에서 집행될수 있도록 집행중에 있는 콤퓨터프로그람에 대한 절대주소를 할당하는 처리과정

디스크배정표 Disk Allocation Table

2차기억기의 블로크들이 자유상태에 있고 파일들에 배정하기 위해 사용하라는것을 가리키는 표

디스크캐쉬 Disk Cache

보통 주기억기에 있으면서 디스크기억기와 주기억기의 예비구역사이에서 디스크블로크의 캐쉬로서의 기능을 수행하는 완충기

대칭다중처리 Symmetric Multiprocessing (SMP)

조작체계가 임의의 사용가능한 처리기 또는 여러개의 사용가능한 처리기들에서 동시에 집행되도록 하는 다중처리의 형태

대화(조종) Session

단일대화사용자응용 또는 조작체계기능을 표현하는 한개이상의 프로쎄스들의 집합. 모든 건반 및 마우스입력은 전경대화로 지정되며 전경대화로부터의 모든 출력은 현시화면으로 출구된다.

론리주소 Logical Address

기억기에 대한 현재의 할당과 독립적인 기억기위치의 참조. 기억기접근을 하기전에 물리주소로 변환해야 한다.

론리레코드 Logical Record

물리적환경과 독립적인 레코드. 한개의 론리레코드의 일부분은 각이한 물리레코드들을 차지할수 있고 또는 몇개의 론리레코드들이나 론리레코드들의 일부분들은 한개의 물리레코드를 차지할수 있다.

림계구간 Critical Section

콤퓨터프로그람의 비동기수속에서 다른 비동기수속의 관련된 림계구간을 사용하여 동

시에 집행할수 없는 부분, 호상배제를 참고

레코드 Record

하나의 단위로 취급되는 자료요소들의 그룹

마당 Field

(1)레코드의 한 부분으로 정의된 론리자료, (2)자료항목, 자료집합, 지시자 또는 련결자를 포함할수 있는 레코드의 요소단위

마크로핵심부 Macrokernel

넓은 범위의 봉사들을 제공하는 대규모조작체계의 핵

마이크로핵심부 Microkernel

프로쎄스일정작성, 기억기관리, 통신봉사들을 제공하며 다른 프로쎄스들에 의거하여 조작체계핵심부와 전통적으로 관련이 있는 몇가지 기능들을 수행하는 소형의 특권조작체계의 핵

망조작체계 Network Operating System

콤퓨터망에서 공통봉사기체계를 사용할수 있게 하는 조작체계에 추가적인 쏘프트웨어

명중률 Hit Ratio

2준위기억기에서 즉 고속기억기(즉 캐쉬)에서 발견되는 모든 기억기접근들의 비률

무리일정작성법 Gang Scheduling

동일한 시간에 1 대 1에 기초하여 처리기들의 모임에서 관련된 스레드들의 모임을 실행시키기 위한 일정작성법

물리주소 Physical Address

기억기의 자료단위(실례로 주기억기의 단어 또는 바이트, 2차기억기의 블로크)의 절대위치

미리페지화 Prepaging

페지부재에 의하여 요구된것이 아닌 다른 페지들의 검색. 이것은 가까운 시간에 추가적인 페지들이 요구되면 디스크입출력을 보존할것을 미리 예견한것이다. 요구페지화와 비교해 보라.

믿음직한 체계 Trusted System

주어 진 보안방책을 실현하기 위하여 확증할수 있는 콤퓨터와 조작체계

방식절환 Mode Switch

처리기가 다른 방식(핵심부 또는 프로쎄스)에서 집행하도록 하게 하는 하드웨어조작, 프로쎄스에서 핵심부로 방식을 절환할 때 프로그람계수기와 처리기상태단어, 다른 등록기들이 보관된다. 방식을 핵심부로부터 프로쎄스로 절환하면 정보는 회복된다.

변환미리보기완충기 Translation Lookaside Buffer (TLB)

페지식가상기억기기구의 부분으로서 최근에 참조된 페지표입구점들을 유지하는데 사용되는 고속캐쉬. TLB는 페지표입구점들을 검색하기 위한 주기억기에로의 접근빈도를 줄인다.

병행 Concurrent

공통적인 시간간격에서 발생하는 프로쎄스들이나 스레드들에 관한것으로서 그것들은 공동자원들을 교대로 공유할수 있다.

봉사기 Server

(1)통보문을 통하여 의뢰기에서 오는 요청에 응답하는 프로쎄스, (2) 망에서 다른 국들에 기능들을 제공하는 자료국. 실례로서 파일봉사기, 인쇄봉사기, 우편봉사기를 들수 있다.

분산조작체계 Distributed Operating System

콤퓨터들의 망에 의하여 공유되는 일반조작체계, 분산조작체계는 프로쎄스간 통신, 프로쎄스이주, 호상배제 및 교착의 예방과 검출에 대한 지원을 제공한다.

불균일기억기접근다중처리기 Nonuniform memory Access (NUMA) Multiprocessor

주어 진 처리기로부터 기억기단어에로의 호출시간이 기억기단어의 위치에 따라서 변하는 공유기억기형다중처리기

블로크 Block

(1) 어떤 단위로 기록되는 련속된 레코드들의 집합으로서 그 단위들은 블로크사이에 있는 공백으로 구분된다. (2) 어떤 단위로 송신되는 비트들의 그룹

비동기조작 Asynchronous Operation

명시된 사건에 대하여 규칙적인 또는 예측가능한 시간관계가 없이 임의로 발생하는 조작, 실례로 콤퓨터프로그람의 집행시 임의의 순간에 조종을 받을수 있는 오유진단루틴의 호출을 들수 있다.

비루스 Virus

쓸모 있는 프로그람안에 매몰된 실증되지 않은 비밀루틴. 프로그람집행은 비밀루틴의 집행을 초래한다.

비법쏘프트웨어 Malicious Software

목표로 삼은 콤퓨터에 위험을 발생시키거나 자원들을 다 써버리도록 설계한 쏘프트웨어. 비법적인 쏘프트웨어(멀웨어)는 흔히 합법적인 쏘프트웨어안에 숨겨 져 있거나 합법적인 쏘프트웨어인것처럼 가장한다. 어떤 경우에는 전자우편이나 감염된 플로피디스크를 거쳐 다른 콤퓨터들에로 그자체를 퍼뜨린다. 비법적인 쏘프트웨어의 형태에는 비루스들, 트로이목마들, 웜들 그리고 봉사거절공격을 개시하기 위한 은폐쏘프트웨어들이 있다.

비특권상태 Nonprivileged State

정지명령과 입출력명령들과 같이 중요한 하드웨어명령들을 집행할수 없게 하는 집행상태

배분 Dispatch*

실행준비가 되여 있는 일감이나 과제에 처리기시간을 배정하는것

베어울프 Beowulf

그것이 구축되여 있는 동안 계산작업을 수행하는 능력을 손상시킴이 없이 총체적인 체계의 가격 대 성능비를 최소로 하는데 중점을 둔 클라스터식계산클라스를 정의한다. 대부분의 베어울프체계들은 LINUX 콤퓨터들에서 실현되고 있다.

사슬목록 Chained List

자료항목들은 분산될수 있지만 매개 항목은 다음항목을 지정하기 위한 식별자를 포함하는 목록

상대주소 Relative Address

기준주소로부터의 변위로 계산되는 주소

상봉 Rendezvous

통보문넘기기에서 통보문송신자와 수신자가 모두 통보문이 전달될 때까지 폐색되는 상태

상주모임 Resident Set

주어 진 시간에 주기억기에 실제로 존재하는 프로쎄스의 부분. 작업모임과 비교할것

선취 Preemption

프로쎄스가 자원의 사용을 끝내기전에 그 프로쎄스로부터 자원을 회수하는것

소비되는 자원 Consumable Resource

창조(산생)되고 파괴(소비)되는 자원, 프로쎄스가 자원을 획득하면 자원은 존재를 마친다. 소비자원의 실례로서 새치기들, 신호들 통보문들 그리고 입출력완충기들의 정보를 들수 있다.

생명폐쇄 Livelock

어떤 유효작업을 함이 없이 다른 프로쎄스(들)의 변화에 응답하여 두개 또는 그이상의 프로쎄스들이 련속적으로 자기들의 상태를 변화시키는것. 이것은 진척이 되지 않는 점에서는 교착과 류사하지만 프로쎄스가 폐색되지 않거나 무엇인가를 기다리는 점에서는 교착과 차이 난다.

순차파일 Sequential File

레코드들이 한개 또는 그이상의 열쇠마당들의 값에 따라 배렬되고 파일의 시작부터 같은 순서로 처리되는 파일

순차호출 Sequential Access

자료가 배렬된것과 같은 순서로 기억장치 또는 자료매체에 자료를 입구시키거나 자료가 입구된것과 같은 순서로 자료를 획득하는 능력

스핀폐쇄 Spin Lock

프로쎄스가 사용성을 가리키는 자물쇠변수의 값을 기다리는 무한고리에서 집행하도록 하는 호상배제기구

시간공유 Time Sharing

여러 사용자들이 장치를 동시에 사용하는것

시간세분 Time Slicing

두개 또는 그이상의 프로쎄스들이 같은 처리기에서 시간량자를 할당받는 조작방식

신호기 Semaphore

프로쎄스들사이에서 신호하기 위하여 사용되는 옹근수값. 신호기에 대하여 3가지 조작만을 수행할수 있다. 이러한 조작들 즉 초기화, 감소, 증가는 모두 원자적이다. 신호기에 대한 정확한 정의에 따르면 감소조작은 프로쎄스의 폐색을 초래할수 있으며 증가조작은 프로쎄스의 비폐색을 초래할수 있다.

실시간과제 Real-Time Task

어떤 프로쎄스나 기능 또는 콤퓨터체계밖에서 일어 나는 사건모임과 관련하여 집행되며 외부환경과 효률적으로 그리고 정확히 대화하기 위하여 한개 또는 그이상의 기한

부를 만족시켜야 하는 과제

실시간체계 Real-Time System

실시간과제들을 일정작성하고 관리하여야 하는 조작체계

새치기 Interrupt

콤퓨터프로그람의 집행과 같은 프로쎄스의 중지. 이것은 그 프로쎄스밖에서 일어나는 사건에 의하여 발생하여 프로쎄스가 회복될수 있는것과 같은 방법으로 수행된다.

새치기처리기 Interrupt Handller

일반적으로 조작체계의 한 부분인 루틴. 새치기가 발생하면 조종은 해당한 새치기조종기에로 넘어 가 새치기를 일으킨 조건에 응답하여 어떤 작업을 한다.

색인 달린 순차파일 Indexed Sequential File

레코드들이 열쇠마당의 값들에 따라 배렬되여 있는 파일. 주파일은 열쇠값들의 부분목록을 포함하는 색인파일로서 추가된다. 색인은 요구되는 레코드의 부근에 재빨리 도달하기 위한 표찾기능력을 제공한다.

색인 달린 파일 Indexed File

레코드들이 열쇠마당의 값에 따라서 접근되는 파일. 색인은 매개 열쇠값에 기초하여 매개 레코드의 위치를 지적하는데 필요하다.

색인에 의한 순차접근 Indexed Sequential Access*

임의로 분할된 순차파일들에 보관되여 있는 열쇠들의 색인을 통하여 기억기구조의 레코드들을 조직하고 접근하는것.

색인에 의한 접근 Indexed Access*

기억된 레코드들의 위치들에 대한 개별적인 색인을 통하여 기억기구조의 레코드들을 조직하고 접근하는것

세균 Bacteria

자기자체를 복제하여 체계자원을 소비하는 프로그람

쉘 Shell

대화사용자지령들과 일감조종언어지령들을 해석하는 조작체계의 부분. 사용자와 조작체계사이의 대면부로서의 기능을 수행한다.

자료기지 Database

한개 또는 그이상의 응용들을 봉사하기 위한 방안에 따라 조직된 보통 조종여유를 가진 호상관련된 자료의 집합. 자료는 그것의 구조나 조직에 관계없이 각이한 프로그람들에 사용할수 있도록 보관된다. 공통적인 방법은 새로운 자료를 추가하고 현존하는 자료를 변경하고 검색하는것이다.

작업모임 Working Set

가상시간 t에서 프로쎄스에 대한 파라메터 Δ를 가진 작업모임 $W(t, \Delta)$는 마지막 Δ 시간단위들에서 참조된 프로쎄스의 페지들의 모임이다.

장치구동프로그람 Device Driver

장치나 입출력모듈을 직접 취급하는 조작체계의 모듈(보통 핵심부에 있다.)

접근방법 Access Method

파일, 레코드 또는 레코드모임을 찾는데 사용하는 방법

조작체계 Operating System

프로그람들의 집행을 조종하고 자원배정, 일정작성, 입출력조종, 자료관리와 같은 봉사들을 제공하는 쏘프트웨어

조이기 Compaction

기억기가 가변크기의 분할구역들로 나눌 때 사용되는 수법. 때때로 조작체계는 분할구역들이 서로 린접하여 하나의 블로크에 모이도록 그것들을 조인다.

주기억기 Main Memeory

콤퓨터체계의 내부에 있으면서 프로그람을 주소화할수 있는 기억기. 그것의 내용은 련속적인 집행이나 처리를 위하여 등록기들에 적재할수 있다.

주소공간 Address Space

콤퓨터프로그람에 사용할수 있는 주소범위

주소변환기 Address Translate

가상주소를 실제주소로 변환하는 기능장치

직접접근 Direct Access

자료의 물리적세포를 가리키는 주소에 의하여 그것들의 상대적인 위치와는 독립적인 순서로 기억장치로부터 자료를 읽어 내거나 기억장치에로 자료를 써 넣는 능력

직접기억기접근 Direct Memory Address(DMA)

DMA 모듈이라고 부르는 전용모듈이 주기억기와 입출력장치사이에서 자료의 교환을 조종하는 입출력의 형태, 처리기는 자료블로크의 이송을 위한 요청을 DMA 모듈에 보내고 블로크가 완전히 이송된 후에야만 새치기된다.

집행문맥 Execution Context

프로쎄스상태와 동일

재사용가능한 자원 Reusable Resource

한번에 오직 한개의 프로쎄스에 의해서만 안전하게 리용될수 있으며 그러한 사용에 의하여 고갈되지 않는 자원. 프로쎄스들은 재사용가능한 자원들을 획득하였다가 다른 프로쎄스들이 다시 사용하도록 해방한다. 재사용가능한 자원들의 실례로서는 처리기들, 입출력통로들, 주기억기와 2차기억기, 장치들 그리고 파일, 자료기지, 신호기들과 같은 자료구조들을 들수 있다.

재진입가능한 수속 Reentrant Procedure

동일한 루틴의 이전의 집행을 끝내기전에 가입하여 정확히 집행할수 있는 루틴

차지기다림 Busy Waiting

일어 날 사건을 기다리는동안 순환코드의 반복되는 집행

클라스터 Cluster

통일적인 자원들로서 함께 작업하는 옹근 콤퓨터들이 호상 련결된 콤퓨터들의 그룹으로서 한대의 콤퓨터라는 착각을 일으킨다. 옹근 콤퓨터라는 용어는 클라스터와 떨어져 자기 혼자서 실행할수 있는 체계를 의미한다.

캐쉬기억기 Cache Memory

주기억기보다 용량은 작고 속도는 높으면서 처리기와 주기억기사이에 놓이는 기억기. 캐쉬는 최근에 사용된 기억기세포들에 대한 완충기로서 동작한다.

탄창 Stack
검색될 다음 자료항목이 목록에 가장 최근에 기억된 항목이 되도록 구축되고 유지되는 목록. 이 방법은 후입선출로 특징 지어 진다.

토막 Segment
가상기억기에서 가상주소를 가지는 블로크. 프로그람의 블로크들은 길이가 서로 다를수도 있고 지어는 동적으로 변할수도 있다.

토막화 Segmentation
프로그람 또는 응용프로그람을 가상기억기의 부분과 같은 토막들로 분할하는것

통보문 Message
통신에 의하여 프로쎄스들사이에서 서로 교환할수 있는 정보블로크

통신구성방식 Communication Architecture
통신기능을 실현하는 하드웨어 및 쏘프트웨어구조

트로이목마 Trojan Horse
쓸모 있는 프로그람내에 매몰된 실증되지 않은 비밀루틴. 이 프로그람의 집행은 비밀루틴의 집행으로 끝난다.

특권명령 Privileged Instruction
특정한 방식에서만 보통은 감시프로그람에 의하여 집행할수 있는 명령

특권상태 Privileged State
모든 하드웨어명령들이 집행되게 허락하는 집행상태

파일 File
하나의 단위로 취급되는 관련된 레코드들의 모임

파일관리체계 File Management System
파일의 사용에서 사용자들과 응용프로그람들에 봉사를 제공하는 체계쏘프트웨어의 모임으로서 파일접근, 등록부유지, 접근조종을 포함한다.

파일배정표 File Allocation Table (FAT)
파일에 배정된 공간에 대한 2차기억기의 물리적위치를 가리키는 표. 매개 파일에는 한개의 파일배정표가 있다.

파일조직 File Organization
파일의 레코드들의 물리적순서로서 파일들을 기억시키고 검색하는데 사용하는 접근방법에 의하여 결정된다.

프로쎄스 Process
집행상태에 있는 프로그람. 프로쎄스는 조작체계에 의하여 조종되고 일정작성된다. 과제와 동일

프로쎄스상태 Process State
조작체계가 프로쎄스를 관리하기 위하여 요구하거나 처리기가 프로쎄스를 정확하게 집행하기 위하여 요구하는 모든 정보. 프로쎄스상태는 프로그람계수기와 자료등록기들과 같은 여러가지 처리기등록기들의 내용을 포함한다. 그것은 또한 프로쎄스의 우선권과 특정한 입출력사건의 완료에 대한 프로쎄스의 기다림 등과 같은 조작체계에서 사

용하기 위한 정보를 포함한다. 집행문맥과 동일

프로쎄스서술자 Process Descriptor

프로쎄스조종블로크와 동일

프로쎄스새끼치기 Process Spawning

다른 프로쎄스에 의한 새로운 프로쎄스의 창조

프로쎄스이주 Process Migration

프로쎄스를 목적기계에서 집행하기 위하여 한 기계에서 다른 기계에로 프로쎄스의 충분한 량의 상태정보를 이송하는것

프로쎄스절환 Process Switch

어떤 프로쎄스에 대한 프로쎄스조종블로크, 등록기들과 다른 정보를 보관하고 그 대신에 다른 프로쎄스의 프로쎄스정보를 교체하여 넣는 방법으로 한 프로쎄스로부터 다른 프로쎄스로 처리기를 절환하는 조작

프로쎄스조종블로크 Process Control Block

조작체계에서 프로쎄스의 명시. 그것은 프로쎄스의 특성지표들과 상태에 대한 정보를 포함하는 자료구조이다.

프로쎄스형태 Process Image

프로그람, 자료, 탄창 및 프로쎄스조종블로크를 포함한 프로쎄스의 모든 구성요소들

프레임 Frame

페지식가상기억기에서 가상기억기의 한개의 페지를 유지하는데 사용되는 주기억기의 고정길이블로크

페지 Page

가상기억기에서 가상주소를 가지고 주기억기와 2차기억기사이에서 하나의 단위로 전송되는 고정길이블로크

페지부재 Page Fault

참조된 단어를 포함하는 페지가 주기억기에 없을 때 발생한다. 이것은 새치기를 일으키며 적당한 페지를 주기억기에로 끌어 들일것을 요구한다.

페지프레임 Page Frame

페지를 유지하는데 사용되는 주기억기의 고정크기의 린접한 블로크

페지화 Paging

주기억기와 2차기억기사이의 페지들의 이송

하쉬법 Hashing

자료의 내용들의 함수로서 주소를 계산하여 자료항목에 대한 기억위치를 선택하는것

하쉬파일 Hash File

레코드들이 열쇠마당의 값들에 따라서 접근되는 파일

함정 Trap

하드웨어에 의하여 자동적으로 활성화되는 규정된 주소에로의 프로그람화되지 않은 조건갈래. 갈래가 일어 나는 시작위치는 기록된다.

함정문 Trap doors

일반적인 접근인증방법들이 없이 접근을 허가하는데 사용되는 프로그람에로의 실증되지 않은 비밀입구점

허락형새치기 Enabled Interrupt

보통 조작체계에 의하여 창조되는것으로서 처리기가 특정한 부류의 새치기요청신호들에 응답하는것

호상배제 Mutual Exclusion

임의의 시간에 프로쎄스들의 모임중에서 오직 하나의 프로쎄스만이 주어 진 자원을 호출할수 있거나 주어 진 기능을 수행할수 있는 그러한 프로쎄스들의 모임이 있는 상태. 림계구간을 볼것

후입선출 Last In First Out (LIFO)

검색해야 할 다음의 항목이 대기렬에 가장 최근에 배치된 항목으로 되는 대기렬수법

흐름관 Pipe

두개의 프로쎄스들이 생산자-소비자모형으로 통신하게 하는 순환완충기. 따라서 그것은 한 프로쎄스에 의하여 씌여 지고 다른 프로쎄스에 의하여 읽어 지는 선입선출대기렬이다. 일부 체계들에서 흐름관은 대기렬의 임의의 항목이 소비를 위하여 선택되도록 일반화된다.

핵심부 Kernel

쏘프트웨어의 가장 많이 사용되는 부분들을 포함하는 조작체계의 한 부분. 일반적으로 핵심부는 항상 주기억기에 보관된다. 핵심부는 특권방식에서 실행되며 프로쎄스의 호출에 응답하며 장치들에 의해 새치기된다.

암호화 Encryption

가역적인 수학적계산에 의하여 평문이나 자료를 리해하기 힘든 형태로 변환하는것

약한 신호기 Weak Semaphore

같은 신호기를 기다리는 모든 프로쎄스들이 규정되지 않은 순서(실례로 알려 지지 않은 순서 또는 애매한 순서)로 집행하게 되는 신호기

요구페지화 Demand Paging

필요한 순간에 2차기억기로부터 주기억기에로 파일의 이송. 미리페지화와 비교할것.

우편함 Mailbox

통보문용대기렬로 사용되는 많은 프로쎄스들사이에서 공유되는 자료구조. 통보문들은 송신기로부터 수신기에로 직접 보내는것이 아니라 먼저 우편함에 보내고 우편함으로부터 검색한다.

응답시간 Response Time

자료체계에서 문의통보문의 전송완료순간부터 응답통보문의 수신시작순간까지의 경과시간으로서 문의말단에서 측정된다.

응용프로그람대면부 Application Programming Interface(API)

특정한 조작체계 또는 도형사용자대면부와 호환성 있는 응용프로그람들을 작성하기 위하여 쏘프트웨어개발자들이 사용하는 표준화된 프로그람작성도구들의 서고

2진신호기 Binary Semaphore
값 0과 1을 가지는 신호기

2차기억기 Secondary Memory
디스크와 테프를 비롯하여 콤퓨터체계밖에 위치하는 기억기

일감 Job
어떤 단위로서 실행하기 위해 제품화된 계산걸음들의 모임

일감조종언어 Job Control Language (JCL)*
일감을 식별하거나 조작체계에 대한 그것의 요구사항을 서술하는데 사용되는 일감에서 명령문들을 표현하기 위하여 설계된 문제지향언어

일괄처리 Batch Processing
콤퓨터프로그람묶음을 집행하는 수법으로서 묶음의 매개 프로그람은 다음 프로그람이 개시되기전에 완료된다.

일정작성 Schedule
배분하려는 일감 또는 과제들을 선택하는것. 어떤 조작체계들에서 입출력조작들과 같은 다른 작업단위들도 역시 일정작성될수 있다.

입출력완충 Spooling
주변장치와 콤퓨터처리기들사이에 자료를 이송할 때 처리지연을 줄이기 위하여 완충기억기로서 2차기억기를 리용하는것

외부쪼각화 External Fragmentation
기억기를 기억기에 할당된 자료블로크(즉 주기억기의 토막)에 해당한 가변크기의 분할구역으로 나눌 때 발생한다. 토막을 기억기의 안팎으로 이동할 때 기억기의 차지된 분할구역들사이에 간격들이 생긴다.

원격수속호출 Remote Procedure Call(RPC)
서로 다른 기계에서 동작하는 두개의 프로그람이 수속호출/복귀문장 및 의미론들을 리용하여 서로 대화하게 하는 수법. 호출된 프로그람과 호출하는 프로그람들은 모두 마치 동료프로그람이 같은 기계에서 실행하고 있는듯이 동작한다.

웜 Worm
망련결을 통하여 콤퓨터들사이를 려행할수 있는 프로그람. 비루스 또는 박테리아를 포함할수 있다.

의뢰기 Client
봉사기프로쎄스들에 통보문을 보내여 봉사들을 요청하는 프로쎄스

참 고 문 헌

ACM Association for Computing Machinery

IEEE Institute of Electrical and Electronics Engineers

IRE Institute of Radio Engineers

ABRA87 Abrams, M., and Podell, H. *Computer and Network Security.* Los Alamitos, CA: IEEE Computer Society Press, 1987.

ADAM92 Adam, J. "Virus Threats and Countermeasures." *IEEE Spectrum,* August 1992.

AGAR89 Agarwal, A. *Analysis of Cache Performance for Operating Systems and Multiprogramming.* Boston: Kluwer Academic Publishers, 1989.

ALMA89 Almasi, G., and Gottlieb, A. *Highly Parallel Computing.* Redwood City, CA: Benjamin/Cummings, 1989.

ALVA90 Alvare, A. "How Crackers Crack Passwords or What Passwords to Avoid." *Proceedings, UNIX Security Workshop II,* August 1990.

ANAN92 Ananda, A.; Tay, B.; and Koh, E. "X Survey of Asynchronous Remote Procedure Calls." *Operating Systems Review,* April 1992.

ANDE80 Anderson, J. *Computer Security Threat Monitoring and Surveillance.* Fort Washington, PA: James P. Anderson Co., April 1980.

ANDE89 Anderson, T.; Laxowska, E.; and Levy, H. "The Performance Implications of Thread Management Alternatives for Shared-Memory Multiprocessors." *IEEE Transactions on Computers,* December 1989.

ANDE92 Anderson, T.; Bershad, B.; Lazowska, E.; and Levy, H. "Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism." *ACM Transactions on Computer Systems,* February 1992.

ANDE97 Anderson, T.; Bershad, B.; Lazowska, E.; and Levy, H. "Thread Management for Shared-Memory Multiprocessors." In [TUCK97].

ANDR90 Andrianoff, S. "A Module on Distributed Systems for the Operating System Course." *Proceedings, Twenty-First SIGCSE Technical Symposium on Computer Science Education, SIGSCE Bulletin,* February 1990.

ARDE80 Arden, B., editor. *What Can Be Automated?* Cambridge, MA: MIT Press, 1980.

ARTS89a Artsy, Y., ed. Special Issue on Process Migration. *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems ,* Winter 1989.

ARTS89b Artsy, Y. "Designing a Process Migration Facility: The Charlotte Experience." *Computer,* September 1989.

ATLA89 Atlas, A., and Blundon, B. "Time to Reach for It All." *UNIX Review,* January 1989.

ATT87a AT&T. *UNIX System Readings and Examples, Volume I.* Englewood Cliffs, NJ: Prentice Hall, 1987.

ATT87b AT&T. *UNIX System Readings and Examples, Volume II.* Englewood Cliffs, NJ: Prentice Hall, 1987.

AXF088 Axford, T. *Concurrent Programming: Fundamental Techniques for Real-Time and Parallel Software Design.* New York: Wiley, 1988.

BACH86 Bach, M. *The Design of the UNIX Operating System.* Englewood Cliffs, NJ: Prentice Hall, 1986.

BACO98 Bacon, J. *Concurrent Systems.* Reading, MA: Addison-Wesley, 1998.

BAEN97 Baentsch, M., et al. "Enhancing the Web's Infrastructure: From Caching to Replication." *Internet Computing,* March/April 1997.

BAER80 Baer, J. *Computer Systems Architecture.* Rockville, MD: Computer Science Press, 1980.

BARB90 Barbosa, V. "Strategies for the Prevention of Communication Deadlocks in Distributed Parallel Programs." *IEEE Transactions on Software Engineering,* November 1990.

BARK89 Barkley, R., and Lee, T. "A Lazy Buddy System Bounded by Two Coalescing Delays per Class." *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles,* December 1989.

BAYS77 Bays, C. "A Comparison of Next-Fit, First-Fit, and Best-Fit." *Communications of the ACM,* March 1977.

BECK90 Beck, L. *System Software.* Reading, MA: Addison-Wesley, 1990.

BECK98 Beck, M., et al. *Linux Kernel Internals.* Reading, MA: Addison-Wesley, 1998.

BELA66 Belady, L. "A Study of Replacement Algorithms for a Virtual Storage Computer." *IBM Systems Journal,* No. 2, 1966.

BEN82 Ben-Ari, M. *Principles of Concurrent Programming.* Englewood Cliffs, NJ: Prentice Hall, 1982.

BEN90 Ben-Ari, M. *Principles of Concurrent and Distributed Programming.* Englewood Cliffs, NJ: Prentice Hall, 1990.

BEN98 Ben-Ari, M., and Burns, A. "Extreme Interleavings." *IEEE Concurrency,*

July-September 1998.

BERN96 Bernstein, P. "Middleware: A Model for Distributed System Services." *Communications of the ACM,* February 1996.

BERS96 Berson, A. *Client/Server Architecture.* New York: McGraw-Hill, 1996.

BIRR89 Bin-ell, A. *An Introduction to Programming with Threads.* SRC Research Report 35, Compaq Systems Research Center, Palo Alto, CA, January 1989. Available at http://www.research. digital.com/SRC.

BLAC90 Black, D. "Scheduling Support for Concurrency and Parallelism in the Mach Operating System." *Computer,* May 1990.

BOEB85 Boebert, W.; Kain, R.; and Young, W. "Secure Computing: the Secure Ada Target Approach." *Scientific Honeyweller,* July 1985. Reprinted in [ABRA87].

BOLL99 Gollmann, D. *Computer Security.* New York: Wiley, 1999.

BOL089 Bolosky, W.; Fitzgerald, R.; and Scott, M. "Simple But Effective Techniques for NUMA Memory Management." *Proceedings, Twelfth ACM Symposium on Operating Systems Principles,* December 1989.

BONW94 Bonwick, J. "An Object-Caching Memory Allocator." *Proceedings, USENIX Summer Technical Conference,* 1994.

BORG90 Borg, A.; Kessler, R.; and Wall, D. "Generation and Analysis of Very Long Address Traces." *Proceedings of the 17th Annual International Symposium on Computer Architecture,* May 1990.

BREN89 Brent, R. "Efficient Implementation of the First-Fit Strategy for Dynamic Storage Allocation." *ACM Transactions on Programming Languages and Systems,* July 1989.

BREW97 Brewer, E. "Clustering: Multiply and Conquer." *Data Communications,* July 1997.

BRIA99 Briand, L, and Roy, D. *Meeting Deadlines in Hard Real-Time Systems: The Rate Monot-onic Approach.* Los Alamitos, CA: IEEE Computer Society Press, 1999.

BRIN73 Brinch-Hansen, P. *Operating System Principles.* Englewood Cliffs, NJ: Prentice Hall, 1973.

BROW84 Brown, R.: Denning, P.; and Tichy, W. "Advanced Operating Systems." *Computer,* October 1984.

BUHR95 Buhr, P., and Fortier, M. "Monitor Classification." *ACM Computing Surveys,* March 1995.

BUTT99 Buttazzo, G. "Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks

in Hard Real-Time Environments." *IEEE Transactions on Computers,* October 1999.

BUYY99a Buyya, R. *High Performance Cluster Computing: Architectures and Systems.* Upper Saddle River, NJ: Prentice Hall, 1999.

BUYY99b Buyya, R. *High Performance Cluster Computing: Programming and Applications.* Upper Saddle River, NJ: Prentice Hall, 1999.

BYNU96 Bynum, B., and Camp, T. "After You, Alfonse: A Mutual Exclusion Toolkit," *Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education,* February 1996.

CABR86 Cabrear, L. "The Influence of Workload on Load Balancing Strategies." *USENIX Conference Proceedings,* Summer 1986.

CA096 Cao, P.; Felten, E.; Karlin, A.; and Li, K. "Implementation and Performance of Integrated Application-Controlled File Caching, Prefetching, and Disk Scheduling." *A CM Transactions on Computer Systems,* November 1996.

CARD97 Card, R.; Dumas, E.; and Mevel, F. *The Linux Kernel Book.* New York: Wiley, 1997.

CARR81 Carr, R., and Hennessey, J. "WSClock—A Simple and Efficient Algorithm for Virtual Memory Management." *Proceedings of the Eighth Symposium on Operating System Principles."* December 1981.

CARR84 Carr, R. *Virtual Memory Management.* Ann Arbor, MI: UMI Research Press, 1984.

CARR89 Carriero, N., and Gelernter, D. "How to Write Parallel Programs: A Guide for the Perplexed." *ACM Computing Surveys,* September 1989.

CASA94 Casavant, T., and Singhal, M. *Distributed Computing Systems.* Los Alamitos, CA: IEEE Computer Society Press, 1994.

CAST92 Castillo, C.; Flanagan, E.; and Wilkinson, N. "Object-Oriented Design and Programming." *AT&T Technical Journal,* November/December 1992.

CHAN85 Chandy, K., and Lamport, L. "Distributed Snapshots: Determining Global States of Distributed Systems." *ACM Transactions on Computer Systems,* February 1985.

CHAN90 Chandras, R. "Distributed Message Passing Operating Systems." *Operating Systems Review,* January 1990.

CHAP97 Chapin, S., and Maccabe, A., eds. "Multiprocessor Operating Systems: Harnessing the Power." special issue of *IEEE Concurrency,* April-June 1997.

CHEN92 Chen, J.; Borg, A.; and Jouppi, N. "A Simulation Based Study of TLB Performance." *Proceedings of the 19th Annual International Symposium on Computer*

Architecture, May 1992.
CHEN94 Chen, P.; Lee, E.; Gibson, G.; Katz, R.; and Patterson, D. "RAID: High-Performance, Reliable Secondary Storage." *ACM Computing Surveys,* June 1994.
CHEN96 Chen, S., and Towsley, D. "A Performance Evaluation of RAID Architectures." *IEEE Transactions on Computers,* October 1996.
CHES97 Chess, D. "The Future of Viruses on the Internet." *Proceedings, Virus Bulletin International Conference,* October 1997.
CHU72 Chu, W., and Opderbeck, H. "The Page Fault Frequency Replacement Algorithm." *Proceedings, Fall Joint Computer Conference,* 1972.
CLAR85 dark, D., and Emer, J. "Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement." *ACM Transactions on Computer Systems,* February 1985.
CLAR98 Clarke, D., and Merusi, D. *System Software Programming: The Way Things Work.* Upper Saddle River, NJ: Prentice Hall, 1998.
COFF71 Coffman, E.; Elphick, M.: and Shoshani, A. "System Deadlocks." *Computing Surveys,* June 1971.
COME84 Comer, D., and Fossum, T. *Operating System Design: The Xinu Approach.* Englewood Cliffs, NJ: Prentice Hall, 1984.
COMP98 Compaq Corp. Guide to DECthreads. December 1998. Available at http ://www.appstate.edu/vms_doc/72final/6493/6101pro_contents.html.
CONW63 Conway, M. "Design of a Separable Transition-Diagram Compiler." *Communications oftheACM,]u¥y* 1963.
CONW67 Conway, R.: Maxwell, W.; and Miller, L. *Theory of Scheduling.* Reading, MA: Addison-Wesley, 1967.
COOP89 Cooper, J. *Computer and Communications Security: Strategies for the 1990s.* New York: McGraw-Hill, 1990.
CORB62 Corbato, F.; Merwin-Daggett, M.; and Dealey, R. "An Experimental Time-Sharing System." *Proceedings of the 1962 Spring Joint Computer Conference,* 1962.
CORB68 Corbato, F. "A Paging Experiment with the Multics System." *MIT Project MAC Report MAC-M-384,* May 1968.
CORB96 Corbett, J. "Evaluating Deadlock Detection Methods for Concurrent Software." *IEEE Transactions on Software Engineering,* March 1996.

COX89 Cox, A., and Fowler, R. "The Implementation of a Coherent Memory Abstraction on a NUMA Multiprocessor: Experiences with PLATINUM." *Proceedings, Twelfth ACM Symposium on Operating Systems Principles,* December 1989.

CROW97 Crowley, C. *Operating Systems: A Design-Oriented Approach.* Chicago: Irwin, 1997.

CUST93 Custer, H. *Inside the Windows NT.* Redmond, WA: Microsoft Press, 1993.

CUST94 Custer, H. *Inside the Windows NT File System.* Redmond, WA: Microsoft Press, 1994.

DALE68 Daley, R, and Dennis, R. "Virtual Memory, Processes, and Sharing in MULTICS." *Communications of the ACM,* May 1968.

DALT96 Dalton, W., et al. *Windows NT Server 4: Security, Troubleshooting, and Optimization.* Indianapolis, IN: New Riders Publishing, 1996.

DASG92 Dasgupta, P.; et. al. "The Clouds Distributed Operating System." *IEEE Computer,* November 1992.

DATT90 Datta, A., and Ghosh, S. "Deadlock Detection in Distributed Systems." *Proceedings, Phoenix Conference on Computers and Communications,* March 1990.

DATT92 Datta, A.; Javagal, R.; and Ghosh, S. "An Algorithm for Resource Deadlock Detection In Distributed Systems," *Computer Systems Science and Engineering,* October 1992.

DAVI90 Davies, G., and Burns, A. "The Teaching Language Pascal-FC," *The Computer Journal,* February 1990.

DELL00 Dekker, E., and Newcomer, J. *Developing Windows NT Device Drivers: A Programmer's Handbook.* Reading, MA: Addison Wesley, 2000.

DENN68 Denning, P. "The Working Set Model for Program Behavior." *Communications of the ACM.* May 1968.

DENN70 Denning, P. "Virtual Memory." *Computing Surveys,* September 1970.

DENN80a Denning, P.; Buzen, J.; Dennis, J.: Gaines, R.; Hansen, P.; Lynch, W.; and Organick, E. "Operating Systems." in [ARDE80],

DENN80b Denning, P. "Working Sets Past and Present." *IEEE Transactions on Software Engineering,* January 1980.

DENN84 Denning, P., and Brown, R. "Operating Systems." *Scientific American,* September 1984.

DENN87 Denning, D. "An Intrusion-Detection Model." *IEEE Transactions on Software*

Engineering, February 1987.

DENN90 Denning, P. *Computers Under Attack: Intruders, Worms, and Viruses.* Reading, MA: Addison-Wesley, 1990.

DIJK65 Dijkstra. E. *Cooperating Sequential Processes.* Technological University, Eindhoven, The Netherlands, 1965. (Reprinted in *Great Papers in Computer Science, P.* Laplante, ed.. IEEE Press, New York, NY, 1996).

DIMT98 Dimitoglou. G. "Deadlocks and Methods for Their Detection. Prevention, and Recovery in Modern Operating Systems." *Operating Systems Review.* July 1998.

DOUG89 Douglas. F., and Ousterhout, J. "Process Migration in Sprite: A Status Report." *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems,* Winter 1989.

DOUG91 Douglas. F., and Ousterhout, J. "Transparent Process Migration: Design Alternatives and the Sprite Implementation." *Software Practice and Experience.* August 1991.

DOWD93 Dowdy, L.. and Lowcry, C. *P.S. to Operating Systems.* Upper Saddle River. NJ: Prentice Hall, 1993.

DUBE98 Dube, R. *A Comparison of the Memory Management Sub-Systems in I-'reeBSD and Linux.* Technical Report CS-TR-3929, University of Maryland, September 25, 1998.

EAGE86 Eager, D.; Lazowska. E.; and Zahnorjan, J. "Adaptive Load Sharing in Homogeneous Distributed Systems." *IEEE Transactions on Software Engineering,* May 1986.

ECKE95 Eckerson, W. "Client Server Architecture." *Network World Collaboration,* Winter 1995.

EFF98 Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design.* Sebastopol, CA: O'Reilly, 1998

ENGE80 Enger, N., and Howerton, P. *Computer Security.* New York: Amacom, 1980.

ESKI90 Eskicioglu, M. "Design Issues of Process Migration Facilities in Distributed Systems." *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems and Application Environments,* Summer 1990.

FEIT90a Feitelson, D., and Rudolph. L. "Distributed Hierarchical Control for Parallel Processing." *Computer,* May 1990.

FEIT90b Feitelson, D., and Rudolph, L. "Mapping and Scheduling in a Shared Parallel Environment Using Distributed Hierarchical Control." *Proceedings, 1990 international*

Conference on Parallel Processing, August 1990.

FERR83 Ferrari. D.. and Yih. Y. "VSWS: The Variable-Interval Sampled Working Set Policy." *IEEE Transactions on Software Engineering.* May 1983.

FIDG96 Fidge, C. "Fundamentals of Distributed System Observation." *IEEE Software,* November 1996.

FINK88 Finkel, R. *An Operating Systems Vade Mecum.* Englewood Cliffs. NJ: Prentice Hall. 1988.

FINK89 Finkel, R. "The Process Migration Mechanism of Charlotte." *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems,* Winter 1989.

FINK97 Finkel, R. "What is an Operating System." In [TUCK97].

FLYN72 Flynn, M. "Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers,* September 1972.

FOLK98 Folk, M., and Zoellick, B. *File Structures: An Object-Oriented Approach with C++.* Reading, MA: Addison-Wesley, 1998.

FRAN97 Franz, M. "Dynamic Linking of Software Components." *Computer,* March 1997.

FRIE96 Friedman, M. "RAID Keeps Going and Going and . . ." *IEEE Spectrum.* April 1996.

GALL00 Galli, D. *Distributed Operating Systems: Concepts and Practice.* Upper Saddle River. NJ: Prentice Hall, 2000.

GANA9S Ganapathy, N.. and Schimmel, C. "General Purpose Operating System Support for Multiple Page Sizes." *Proceedings. USENIX Symposium,* 1998.

GARG96 Garg, V. *Principles of Distributed Systems.* Boston: Kluwer Academic Publishers, 1996.

GEHR87 Gehringer, E.; Siewiorek, D.; and Segall, Z. *Parallel Procesing: The Cm* Experience.* Bedford. MA: Digital Press, 1987.

GIBB87 Gibbons, P. "A Stub Generator for Multilanguage RPC in Heterogeneous Environments." *IEEE Transactions on Software Engineering,* January 1987.

GING90 Gingras, A. "Dining Philosophers Revisited." *ACM SIGCSE Bulletin,* September 1990.

GOLD89 Goldman, P. "Mac VM Revealed." *Byte,* September 1989.

GOOD94 Goodheart, B., and Cox, J. *The Magic Garden Explained: The Internals of UNIX System V Release 4.* Englewood Cliffs, NJ: Prentice Hall, 1994.

GOPA85 Gopal, 1. "Prevention of Store-and-Forward Deadlock in Computer Networks." *IEEE Transactions on Communications,* December 1985.

GOYE99 Goyeneche, J., and Souse, E. "Loadable Kernel Modules." *IEEE Software.*

January/ February 1999.

GRAH95 Graham, J. *Solaris 2.x: Internals and Architecture.* New York: McGraw-Hill. 1995.

GRAY97 Gray. J. *Interprocess Communications in UNIX: The Nooks and Crannies.* Upper Saddle River, NJ: Prentice Hall. 1997.

GROS86 Grosshans, D. *File Systems: Design and Implementation.* Englewood Cliffs, NJ: Prentice Hall, 1986.

GUPT78 Gupta, R., and Franklin, M. "Working Set and Page Fault Frequency Replacement Algorithms: A Performance Comparison." *IEEE Transactions on Computers,* August 1978.

GUYN88 Guynes, J. "Impact of System Response Time on State Anxiety." *Communications of the AC M,* March 1988.

HALD91 Haldar, S., and Subramanian, D. "Fairness in Processor Scheduling in Time Sharing Systems" *Operating Systems Review,* January 1991.

HART97 Hartig, H., et al. "The Performance of a [ˆ-Kernel-Based System." *Proceedings, Sixteenth ACM Symposium on Operating Systems Principles,* December 1997.

HATF72 Hatfield, D. "Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance." *IBM Journal of Research and Development,* January 1972.

HENN96 Hennessy, J., and Patterson, D. *Computer Architecture: A Quantitative Approach.* San Mateo, CA: Morgan Kaufmann, 1996.

HENR84 Henry, G. "The Fair Share Scheduler." *AT&T Bell Laboratories Technical Journal,* October 1984.

HERL90 Herlihy, M. "A Methodology for Implementing Highly Concurrent Data Structures," *Proceedings of the Second ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming,* March 1990.

HOAR74 Hoare, C. "Monitors: An Operating System Structuring Concept." *Communications of the ACM,* October 1974.

HOAR85 Hoare, C. *Communicating Sequential Processes.* Englewood Cliffs, NJ: Prentice Hall, 1985.

HOFF90 Hoffman. L., editor. *Rogue Programs: Viruses, Worms, and Trojan Horses.* New York: Van Nostrand Reinhold, 1990.

HOFR90 Horn, M. "Proof of a Mutual Exclusion Algorithm." *Operating Systems Review,* January 1990.

HOLT72 Holt, R. "Some Deadlock Properties of Computer Systems." *Computing Surveys,* September 1972.

HONG89 Hong, J.; Tan, X.; and Towsley, D. "A Performance Analysis of Minimum Laxity and Earliest Deadline Scheduling in a Real-Time System." *IEEE Transactions on Computers,* December 1989.

HP96 Hewlett Packard. *While Paper on Clustering.* Available at http://www.hp.com/netserver/ partners/papers/wp_clust_696.html, June 1996.

HUCK83 Huck, T. *Comparative Analysis of Computer Architectures.* Stanford University Technical Report Number 83-243, May 1983.

HUCK93 Huck, J., and Hays, J. "Architectural Support for Translation Table Management in Large Address Space Machines." *Proceedings of the 20th Annual International Symposium on Computer Architecture,* May 1993.

HWAN93 Hwang, K. *Advanced Computer Architecture.* New York: McGraw-Hill, 1993.

HWAN99 Hwang, K, et al. "Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space." *IEEE Concurrency,* January-March 1999.

HYMA66 Hyman. H. "Comments on a Problem in Concurrent Programming Control." *Communications of the ACM,* January 1966.

IBM86 IBM National Technical Support, Large Systems. Multiple Virtual Storage (MVS) Virtual Storage Tuning Cookbook. Dallas Systems Center Technical Bulletin G320-0597, June 1986.

1SLO80 Isloor. S., and Marsland, T. "The Deadlock Problem: An Overview." *Computer.* September 1980.

JACO98a Jacob, B., and Mudge, T. "Virtual Memory: Issues of Implementation." *Computer,* June 1998.

JACO98b Jacob. B.. and Mudge, T. "Virtual Memory in Contemporary Microprocessors." *IEEE Micro.* August 1998.

JOHN91 Johnston, B.; Javagal, R.; Datta, A.: and Ghosh. S. "A Distributed Algorithm for Resource Deadlock Detection." *Proceedings, Tenth Annual Phoenix Conference on Computers and Communications.* March 1991.

JOHN92 Johnson, T., and Davis. T. "Space Efficient Parallel Buddy Memory Management." *Proceedings, Third International Conference on Computers and Information.* May 1992.

JONE80 Jones. S., and Schwarz, P. "Experience Using Multiprocrressor Systems—A Status Report." *Computing Surveys,* June 1980.

JUL88 Jul, E.: Levy, H.: Hutchinson. N.; and Black, A. "Fine-Grained Mobility in the Emerald System." *ACM Transactions on Computer Systems.* February 1988.

JUL89 Jul, E. "Migration of Light-Weight Processes in Emerald." *Newsletter of the IEEE Computer Society Technical Committee on Operating Systems* , Winter 1989,

KABA99 Kabay, M. "Annual Virus Survey: Clear and Prevalent Danger." Information Security, August 1999.

KANG98 Kang, S., and Lee, J. "Analysis and Solution of Non-Preemptive Policies for Scheduling Readers and Writers." *Operating Systems Review,* July 1998.

KAPP00 Kapp, C. "Managing Cluster Computers." *Dr. Dobb's Journal,* July 2000.

KATZ89 Katz, R.; Gibson, G.; and Patlerson, D. "Disk System Architecture for High Performance Computing." *Proceedings of the IEEE,* December 1989.

KAY88 K.ay, J., and Lauder. P. "A Fair Share Scheduler." *Communications of the ACM,* January 1988.

K.EPH97a Kephart, J.; Sorkin, G.; Chess, D.: and White, S. "Fighting Computer Viruses." *Scientific American,* November 1997.

KEPH97b Kephart, J.: Sorkin, G.: Swimmer, B.; and White, S. "Blueprint for a Computer Immune System." *Proceedings, Virus Bulletin International Conference,* October 1997.

KESS92 Kessler, R., and Hill, M. "Page Placement Algorithms for Large Real-Indexed Caches." *ACM Transactions on Computer Systems,* November 1992.

KHAL93 Khalidi, Y.: Talluri, M.: Williams, D.: and Nelson, M. "Virtual Memory Support for Multiple Page Sizes." *Proceedings, Fourth Workshop on Workstation Operating Systems,* October 1993.

KHAL96 Khalidi, Y., et al. "Solaris MC: A Multicomputer OS." Proceedings, 1996 USENIX Conference. January 1996.

KILB62 Kilburn, T.; Edwards, D.: Lanigan, M.; and Sumner, F. "One-Level Storage System." *IRE Transactions.* April 1962.

KLEI76 Kleinrock, L. *Queuing Systems, Volume II: Computer Applications.* New York: Wiley. 1976.

KLEI90 Klein, D. "Foiling the Cracker: A Survey of. and Improvements to. Password Security." *Proceedings, UNIX Security Workshop II,* August 1990.

KLEI95 Kleiman, S. "Interrupts as Threads." *Operating System Review,* April 1995.

KLE196 Kleiman. S.; Shah, D.; and Smallders, B. *Programming with Threads.* Upper Saddle River. NJ: Prentice Hall, 1996.

KNUT71 Knuth, D. "An Experimental Study of FORTRAN Programs." *Software Practice and Experience.* Vol. 1, 1971.

KNUT97 Knuth, D. *The Art of Computer Programming, Volume 1: Fundamental Algorithms.* Reading, MA: Addison-Wesley, 1997.

KRIS94 Krishna, C, and Lee, Y., eds. "Special Issue on Real-Time Systems." *Proceedings of the IEEE,* January 1994.

KRON90 Kron, P. "A Software Developer Looks at OS/2." *Byte,* August 1990.

LAMP74 Lamport, L. "A New Solution to Dijkstra's Concurrent Programming Problem." *Communications oflheACM,* August 1974.

LAMP78 Lamport, L. "Time, Clocks, and the Ordering of Events in a Distributed System." *Communications of the ACM,* July 1978.

LAMP80 Lampson, B., and Redell D. "Experience with Proceses and Monitors in Mesa." *Communications of the ACM,* February 1980.

LAMP86 Lamport, L. "The Mutual Exclusion Problem." *Journal of the ACM,* April 1986.

LAMP91 Lamport, L. "The Mutual Exclusion Problem Has Been Solved." *Communications of the ACM,* January 1991.

LARO92 LaRowe, R.; Holliday, M.; and Ellis, C. "An Analysis of Dynamic Page Placement an a NUMA Multiprocessor." Proceedings, 7992 *ACM SIGMETRICS and PERFORMANCE '92,* June 1992.

LEBL87 LeBlanc, T., and Mellor-Crummey, J. "Debugging Parallel Programs with Instant Replay." *IEEE Transactions on Computers,* April 1987.

LEE93 Lee, Y., and Krishna, C., eds. *Readings in Real-Time Systems.* Los Alamitos, CA: IEEE Computer Society Press, 1993.

LELA86 Leiand, W., and Ott, T. "Load-Balancing Heuristics and Process Behavior." *Proceedings, ACM SigMetrics Performance 1986 Conference,* 1986.

LERO76 Leroudier, J., and Potier, D. "Principles of Optimality for Multiprogramming." *Proceedings, International Symposium on Computer Performance Modeling, Measurement, and Evaluation.* March 1976.

LETW88 Letwin, G. *Inside OS/2.* Redmond, WA: Microsoft Press, 1988.

LEUT90 Leutenegger, S., and Vernon, M. "The Performance of Multiprogrammed Multiprocessor Scheduling Policies." *Proceedings, Conference on Measurement and Modeling of Computer Systems,* May 1990.

LEWI96 Lewis, B., and Berg, D. *Threads Primer.* Upper Saddle River, NJ: Prentice Hall, 1996.

LIED95 Liedtke, J. "On u-Kernel Construction." *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles,* December 1995.

LIED96a Liedtke, J. "Toward Real Microkernels." *Communications of the ACM,* September 1996.

LIED96b Liedtke, J. "Microkernels Must and Can Be Small." *Proceedings, Fifth International Workshop on Object Orientation in Operating Systems,* October 1996.

LIST93 Lister, A., and Eager, R. *Fundamentals of Operating Systems.* New York: Springer-Verlag, 1993.

LIU73 Liu, C., and Layland, J. "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment." *Journal of the ACM,* February 1973.

LIVA90 Livadas, P. *File Structures: Theory and Practice.* Englewood Cliffs, NJ: Prentice Hall, 1990.

LYNC96 Lynch, N. *Distributed Algorithms.* San Francisco, CA: Morgan Kaufmann, 1996.

MADS93 Madsen, J. "World Record in Password Checking." *USENET, comp.security.misc newsgroup,* August 18, 1993.

MAEK87 Maekawa, M.; Oldehoeft, A.; and Oldehoeft, R. *Operating Systems: Advanced Concepts.* Menio Park, CA: Benjamin Cummings, 1987.

MAJU88 Majumdar, S.; Eager, D.; and Bunt, R. "Scheduling in Multiprogrammed Parallel *Systems." Proceedings, Conference on Measurement and Modeling of Computer Systems,* May 1988.

MANC00 Mancill, T. "LINUX in the Corporate Network?" *Business Communications Review,* January 2000.

MART88 Martin, J. *Principles of Data Communication.* Englewood Cliffs, NJ: Prentice Hall, 1988.

MASS97 Massiglia, P., editor. *The RAID Book: A Storage System Technology Handbook.* St. Peter, MN: The Raid Advisory Board, 1997.

MCKU96 McKusick, M.; Bostic, K.; Karels, M.; and Ouartermain, J. *The Design and Implementation of the 4.4BSD UNIX Operating System.* Reading, MA: Addison-Wesley, 1996.

MEE96a Mee, C., and Daniel, E. eds. *Magnetic Recording Technology.* New York: McGraw Hill, 1996.

MEE96b Mee, C., and Daniel, E. eds. *Magnetic Storage Handbook.* New York: McGraw Hill, 1996.

MESS96 Messer, A., and Wilkinson, T. "Components for Operating System Design."

Proceedings, Fifth International Workshop on Object Orientation in Operating Systems, October 1996.

MILE92 Milenkovic, M. *Operating Systems: Concepts and Design.* New York: McGraw-Hill, 1992.

MILO96 Milojicic, D.; Douglis, F.; Paindaveine, Y.; Wheeler, R.; and Zhou, S. *Process Migration.* TOGRITechnicalReport,December1996.
Available at http://www.opengroup.org/-dejan/ papers/index.htm.

MORG92 Morgan, K. "The RTOS Difference." *Byte,* August 1992.

MS96 Microsoft Corp. *Microsoft Windows NT Workstation Resource Kit.* Redmond, WA: Microsoft Press, 1996.

MUKH96 Mukherjee, B., and Karsten, S. "Operating Systems for Parallel Machines." In *Parallel Computers: Theory and Practice.* Edited by T. Casavant, P. Tvrkik, and F. Plasil. Los Alami-tos, CA: IEEE Computer Society Press, 1996.

NACH97 Nachenberg, C. "Computer Virus-Antivirus Coevolution." *Communications of the A CM.* January 1997.

NAGA97 Nagar, R. *Windows NT File System Internals.* Sebastopol, CA: O'Reilly, 1997.

NEHM75 Nehmer, J. "Dispatcher Primitives for the Construction of Operating System Kernels." *Ada Informalica,* vol. 5, 1975.

NELS88 Nelson, M.; Welch, B.; and Ousterhout, J. "Caching in the Sprite Network File System." *A CM Transactions on Computer Systems,* February 1988.

NELS91 Nelson, G. *Systems Programming with Modula-3.* Englewood Cliffs, NJ: Prentice Hall, 1991.

NG98 Ng, S. "Advances in Disk Technology: Performance Issues." *Computer,* May 1989.

NUTT94 Nuttal, M. "A Brief Survey of Systems Providing Process or Object Migration Facilities." *Operating Systems Review,* October 1994.

NUTT00 Nutt, G. *Operating Systems: A Modern Perspective.* Reading, MA: Addison-Wesley, 2000.

OUST85 Ousterhout, J., et al. "A Trace-Drive Analysis of the UNIX 4.2 BSD File System." *Proceedings, Tenth ACM Symposium on Operating System Principles,* 1985.

OUST88 Ousterhout, J., et al. "The Sprite Network Operating System." *Computer,* February 1988.

PAI00 Pai, V.; Druschel, P.; and Zwaenepoel, W. "10-Lite: A Unified I/O Buffering and Caching System." *ACM Transactions on Computer Systems,* February 2000.

PANW88 Panwar, S.; Towsley, D.; and Wolf, J. "Optimal Scheduling Policies for a Class

of Queues with Customer Deadlines in the Beginning of Service." *Journal of the ACM,* October 1988.

PATT82 Patterson, D., and Sequin, C. "A VLSI RISC." *Computer,* September 1982.

PATT85 Patterson, D. "Reduced Instruction Set Computers." *Communications of the ACM, January* 1985.

PATT88 Patterson, D.; Gibson, G.; and Katz, R. "A Case for Redundant Arrays of Inexpensive Disks (RAID)." *Proceedings, ACM SIGMOD Conference of Management of Data,* June 1988.

PATT98 Patterson, D., and Hennessy, J. *Computer Organization and Design: The Hardware/Software Interface.* San Mateo, CA: Morgan Kaufmann, 1998.

PETE77 Peterson, J., and Norman, T. "Buddy Systems." *Communications of the ACM,* June 1977.

PETE81 Peterson, G. "Myths About the Mutual Exclusion Problem." *Information Processing Letters,* June 1981.

PFIS98 Pfister, G. *In Search of Clusters.* Upper Saddle River, NJ: Prentice Hail, 1998.

PFLE97 Pfleeger, C. *Security in Computing.* Upper Saddle River, NJ: Prentice Hall PTR, 1997.

PHAM96 Pham. T., and Garg, P. *Multithreaded Programming with Windows NT.* Upper Saddle River, NJ: Prentice Hall, 1996.

PINK89 Pinkert, J., and Wear. L. *Operating Systems: Concepts. Policies, and Mechanisms.* Engle-wood Cliffs. NJ: Prentice Hall, 1989.

PIZZ89 Pizzarello, A. "Memory Management for a Large Operating System." *Proceedings.International Conference on Measurement and Modeling of Computer Systems.* May 1989.

POPE85 Popek, G.. and Walker, B. *The LOCUS Distributed System Architecture.* Cambridge, MA: MIT Press, 1985.

PORR92 Porras. P. *STA T: A Stale Transition Analysis Tool for Intrusion Detection.* Master's Thesis, University of California at Santa Barbara, July 1992.

PRAM84 Pramanik, S.. and Weinberg, B. "The Implementation Kit with Monitors," *SIGPLAN Notices,* Number 9. 1984.

PRZY88 Przyhylski, S.; Horowit/, M.; and Hennessy, J. "Performance Trade-Offs in Cache Design." Proceedings. Fifteenth Annual International Symposium on Computer Architecture. *June 1988.*

RAJA00 Rajagopal. R. *Introduction to Microsoft Windows NT Cluster Server. Boca Ralon, FL:* CRC Press. 2000.

RAMA94 Ramamritriam, K... and Stankovie, J. "Scheduling Algorithms and Operating Systems Support for Real-Time Systems." *Proceedings of the IEEE,* January 1994.

RASH88 Rashid, R., el al. "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures." *IEEE Transactions on Computers,* August 1988.

RAYN86 Raynal. M. *Algorithms for Mutual Exclusion.* Cambridge, MA: MIT Press. 1986.

RAYN88 Raynal. M. *Distributed Algorithms and Protocols.* New York: Wiley. 1988.

RAYN90 Raynal, M.. and Helary. J. *Synchronization and Control of Distributed Systems and Programs.* New York: Wiley, 1990.

REAG00a Reagan, P. *Client/Server Computing.* Upper Saddle River, NJ: Prentice Hall, 2000.

REAG00b Reagan, P. *Client/Server Network: Design, Operation and Management.* Upper Saddle River. NJ: Prentice Hall, 2000.

RICA81 Ricart, G., and Agrawala, A. "An Optimal Algorithm for Mutual Exclusion in Computer Networks." *Communications of the ACM,* January 1981 (Corrigendum in *Communications of the ACM.* September 1981).

RICA83 Ricart, G., and Agrawala. A. "Author's Response to 'On Mutual Exclusion in Computer Networks' by Carvalho and Roucairol." *Communications of the ACM,* February 1983.

R1CH97 Richter. J. *Advanced Windows.* Redmond, WA: Microsoft Press, 1997.

RIDG97 Ridge. D., et al. "Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs." *Proceedings, fEEE Aerospace,* 1997.

RITC74 Ritchie, D., and Thompson, K. "The UNIX Time-Sharing System." *Communications of the ACM,* July 1974.

RITC78a Rilchie. D., and Thompson, K. "The UNIX Time-Sharing System." *The Bell System Technical Journal,* July-August 1978.

RITC78b Ritchie, D. "UNIX Time-Sharing System: A Retrospective." *The Bell System Technical Journal,* July-August 1978.

RITC84 Ritchie, D. "The Evolution of the UNIX Time-Sharing System." *A T& T Bell laboratories Technical Journal,* October 1984.

ROBI90 Robinson, J., and Devarakonda, M. "Data Cache Management Using Frequency-Based Replacement." *Proceedings, Conference on Measurement and Modeling of Computer Systems,* May 1990.

ROSC00 Rosch, W. *The Winn L. Rosch Hardware Bible.* Indianapolis, IN: Sams, 2000.

ROSE78 Rosenkrantz, D.: Steams, R.; and Lewis, P. "System Level Concurrency Control in Distributed Database Systems." *ACM Transactions on Database Systems,* June 1978.

RUDO90 Rudolph. B. "Self-Assessment Procedure XXI: Concurrency," *Communications of llie A CM.* May 1990.

RUSL99 Rusling, D, *The Linux Kernel.* Available at http://www.linuxdoc.org.

SALU94 Salus, P. "UNIX at 25;' *Byte,* October 1994.

SAND94 Sandhu, R., and Samarati, P. "Access Control: Principles and Practice." *IEEE Communications.* September 1994.

SATY81 Satyanarayanan, M., and Bhandarkar, D. "Design Trade-Offs in VAX-11 Translation Buffer Organization." *Computer,* December 1981.

SAUE81 Sauer, C., and Chandy, K. *Computer Systems Performance Modeling.* Englewood Cliffs. NJ: Prentice Hall. 1981.

SCHA62 Schay. G., and Spruth, W. "Analysis of a File Addressing Method." *Communications of theACM,* August 1962.

SCHI94 Schimmel, C. *UNIX Systems for Modern Architectures.* Reading,MA: Addison- Wesley, 1994.

SCHM97 Schmidt. D. "Distributed Object Computing." *IEEE Communications Magazine,* February 1997.

SCHN96 Schneier, B. *Applied Cryptography.* New York: Wiley, 1996.

SCHN99 Schneier, B. "The Trojan Horse Race." *Communications of the ACM,* September 1999.

SCHW96 Schwaderer, W., and Wilson, A. *Understanding I/O Subsystems.* Milpitas, CA: Adaptec Press, 1996.

SEVC96 Sevcik, P. "Designing a High-Pcrformance Web Site." *Business Communications Review,* March 1996.

SHA91 Sha. L.; Klein. M.: and Goodenough, J. "Rate Monotonic Analysis for Real-Time Systems." in [TILB91].

SHA94 Sha. L.; Rajkumar, R.; and Sathaye, S- "Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems." *Proceedings of the IEEE,* January 1994.

SHEL97 Sheldon, T. *Window. NT Security Handbook.* New York: Osborne Mcgraw-Hill, 1997.

SHNE84 Shneiderman, B. "Response Time and Display Rate in Human Performance with Computers." *ACM Computing Surveys,* September 1984.

SHIV92 Shivaratri, N.; Krueger, P.; and Singhal. M. "Load Distributing for Locally Distributed Systems." *Computer.* December 1992.

SHOR75 Shore, J. "On the External Storage Fragmentation Produced by First-Fit and Best-Fit Allocation Strategies," *Communications of the ACM.* August, 1975.

SHOR97 Short, R.: Gamachc, R.; Vert, J.: and Massa, M. "Windows NT Clusters for Availability and Scalability." *Proceedings, COMPCON Spring 97,* February 1997.

SHUB90 Shub, C. "ACM Forum: Comment on A Self-Assessment Procedure on Operating Systems." *Communications of the ACM,* September 1990.

SIEB83 Sieber, J. *TRIX: A Communications-Oriented Operating System.* Ms.S. Thesis, MIT, September 1983.

SILB98 Silberschatz. A., and Galvin, P. *Operating System Concepts.* Reading, MA: Addison-Wesley. 1998.

SILB00 Silberschatz, A.; Galvin, P.; and Gagne, G. *Applied Operating System Concepts.* Reading, MA: Addison-Wesley, 2000.

SING94a Singhat, M., and Shivaratri, N. *Advanced Concepts in Operating Systems.* New York: McGraw-Hill, 1994.

SING94b Singhal, M. "Deadlock Detection in Distributed Systems." In [CASA94J.

SING99 Singh, H. *Progressing to Distributed Multiprocessing.* Upper Saddle River, NJ: Prentice Hall, 1999.

SINH97 Sinha, P. *Distributed Operating Systems.* Piscataway, NJ: IEEE Press, 1997.

SMIT82 Smith, A. "Cache Memories." *ACM Computing Surveys,* September 1982.

SMIT83 Smith. D. "Faster Is Better: A Business Case for Subsecond Response Time." *Computer-world, Apri¥* 18,1983.

SMIT85 Smith, A. "Disk Cache—Miss Ratio Analysis and Design Considerations." *A CM Transactions on Computer Systems.* August 1985.

SMIT88 Smith, J. "A Survey of Process Migration Mechanisms," *Operating Systems Review.* July 1988.

SMIT89 Smith. J. "Implementing Remote *fork<)* with Checkpoint/restart." *Newsletter of the IEEE Computer Society Technical Commiliee on Operating Systems.* Winter 1989.

SOLO98a Solomon, D. *Inside Windows NT.* Redmond, WA: Microsofl Press. 1998.

SOLO98b Solomon. D. "The Windows NT Kernel Architecture." *Computer.* October.

1998.
SPAF89 Spafford, E.; Heaphy, K.; and Ferbrache. D. *Computer Viruses.* Arlington, VA: ADAPSO, 1989.
SPAF92 Spafford, E. "Observing Reusable Password Choices." *Proceedings, UNIX Security Symposium HI,* September 1992.
STAL98 Stallings. W. *Cryptography and Network Security: Principles and Practice, 2nd ed.* Upper Saddle River. NJ: Prentice Hall, 1995.
STAL00 Stallings, W. *Computer Organization and Architecture,* 5th ed. Upper Saddle River, NJ: Prentice Hall, 2000.
STAN89 Stankovic, J., and Ramamrithan, K. "The Spring Kernel: A New Paradigm for Real-Time Operating Systems." *Operating Systems Review,* July 1989.
STAN93 Stankovic, J., and Ramamritham, K., eds. *Advances in Real-Time Systems.* Los Alami-tos, CA: IEEE Computer Society Press. 1993.
STEE95 Steensgarrd, B., and Jul, E. "Object and Native Code Thread Mobility Among Heterogeneous Computers." *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles,* December 1995.
STEP93 Stephenson. P. "Preventive Medicine." *LAN Magazine,* November 1993.
STER99 Sterling, T., et al. *How to Build a Beowulf.* Cambridge. MA: MIT Press, 1999.
STON93 Stone, H. *High-Performance Computer Architecture.* Reading, MA: Addison-Wesley. 1993.
STRE83 Strecker, W. "Transient Behavior of Cache Memories." *ACM Transactions on Computer Systems.* November 1983.
STUC85 Stuck, B., and Arthurs, E. *A Computer and Communications Network Performance Analysis Primer.* Englewood Cliffs, NJ: Prentice Hall, 1985.
SLJN99 Sun Microsystems. "Sun Cluster Architecture: A White Paper." *Proceedings, IEEE Computer Society International Workshop on Cluster Computing,* December 1999.
SUTT97 Sutton, S. *Windows NT Security Guide.* Reading. MA: Addison-Wesley. 1997.
SUZU82 Suzuki, I., and Kasami. T. "An Optimality Theory for Mutual Exclusion Algorithms in Computer Networks." *Proceedings of the Third International Conference on Distributed Computing Systems.* October 1982.
TAIV96 Taivalsaari. A, "On the Nature of Inheritance." *ACM Computing Surveys,* September 1996.
TALL92 Talluri. M.: Kong, S.; Hill, M.; and Patterson. D. "Tradeoffs in Supporting Two Page Sizes." *Proceedings of the !9th Annual International Symposium on*

Computer Architecture, May 1992.

TAMI83 Tamir, Y., and Sequin, C. "Strategies for Managing the Register File in RISC." *IEEE Transactions on Computers,* November 1983.

TANE78 Tanenbaum. A. "Implications of Structured Programming for Machine Architecture." *Communications of the ACM,* March 1978.

TANE85 Tanenbaum. A.. and Renesse, R. "Distributed Operating Systems." *Computing Surveys.* December 1985.

TANE90 Tanenbaum, A. *Structured Computer Organization.* Englewood Cliffs, NJ: Prentice Hall. 1990.

TANE92 Tanenbaum, A. *Modern Operating Systems.* Englewood Cliffs, NJ: Prentice Hall. 1992.

TANE97 Tanenbaum, A., and Woodhull, A. *Operating Systems: Design and Implementation.* Upper Saddle River, NJ: Prentice Hall, 1997.

TAY90 Tay, B., and Ananda. A. "A Survey of Remote Procedure Calls." *Operating Systems Review,* July 1990.

TEVA87 Tevanian, A., el- al. "Mach Threads and the UNIX Kernel: The Battle for Control." *Proceedings, Summer !987 VSENIX Conference.* June 1987.

THAD81 Thadhani, A. "Interactive User Productivity." *IBM Systems Journal.* No. 1. 1981.

THOM84 Thompson. K. "'Reflections on Trusting Trust (Deliberate Software Bugs)." *Communications of the ACM,* August 1984.

TTLB91 Tilborg, A., and Koob, G., eds. *Foundations of Real-Time Computing: Scheduling and Resource Management.* Boston: Kluwer Academic Publishers, 1991.

TIME90 Time, Inc. *Computer Security, Understanding Computers Series.* Alexandria. VA: Time-Life Books, 1990.

TUCK89 Tucker. A., and Gupta, A. "Process Control and Scheduling Issues for Multipro-grammed Sharcd-Memory Multiprocessors." *Proceedings, Twelfth ACM Symposium on Operating Systems Principles,* December 1989.

TUCK97 Tucker. A., ed. *The Computer Science and Engineering Handbook.* Boca Raton, FL: *CRC Press, 1997.*

VAHA96 Vahalia, U. *UNIX Internals: The New Frontiers.* Upper Saddle River, NJ: Prentice Hall. 1996.

WALK89 Walker, B., and Mathews, R. "Process Migration in AIX's Transparent Computing *Facility."* Newsletter of the IEEE Computer Society Technical Committee on Operating Systems, Winter 1989.

WARD80 Ward. S. "TRIX: A Network-Oriented Operating System." *Proceedings, COMPCON '80,* 1980.

WARR91 Warren, C. -'Rate Monotonic Scheduling." *IEEE Micro,* June 1991.

WAYN94a Wayner, P. "Small Kernels Hit it Big." *Byte,* January 1994.

WAYN94b Wayner, P. "Objects on the March." *Byte.* January 1994.

WEIZ81 Weizer, N. "A History of Operating Systems. "*Datamation,* January 1981.

WEND89 Wendorf, J.; Wendorf, R.: and Tokuda, H. "Scheduling Operating System Processing on Small-Scale Microprocessors." *Proceedings, 22nd Annual Hawaii International Conference on System Science,* January 1989.

WIED87 Wiederhold, G. *File Organization for Database Design.* New York: McGraw-Hill. 1987.

WOOD86 Woodside, C. "Controllability of Computer Performance Tradeoffs Obtained Using Con-trolled-Share Queue Schedulers." *IEEE Transactions on Software Engineering,* October 1986

WOOD89 Woodbury, P. et al. "Shared Memory Multiprocessors: The Right Approach to Parallel Processing." *Proceedings, COMPCON Spring '89,* March 1989.

YOUN87 Young, M., et. al. "The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System." *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles,* December 1987.

ZAHO90 Zahorjan, J., and McCann, C. "Processor Scheduling in Shared Memory Multiprocessors." *Proceedings, Conference on Measurement and Modeling of Computer Systems,* May 1990.

ZAJC93 Zajcew, R., et al. "An OSF/1 UNIX for Massively Parallel Multicomputer*,." *Proceedings, Winter USENIX Conference,* January 1993.

ZEAD97 Zeadally, S. "An Evaluation of the Real-Time Performance of SVR4.0 and SVR4.2." Operating Systems Review, *January 1977.*

색 인

ㄴ

ㄷ

ㅅ

ㅈ

ㅊ

ㅋ

ㅌ

ㅍ

ㅎ

ㅆ

ㅇ

략　　어

API (Application Programming Interface) 응용프로그람작성대면부
CORBA (Common Object Request Broker Architecture) 공통객체요청중개방식
CPU (Central Processing Unit) 중앙처리장치
CTSS (Compatable Time Sharing System) 호환성시분할체계
DES (Data Ecryption Stsndard) 자료암호화표준
DMA (Direct Memory Access) 직접기억기접근
DVD (Digital Versatile Disk) 수자식만능디스크
FAT (File Allocation Table) 파일배정표
FCFS (First Come First Served) 선래선봉사
FIFO (First In First Out) 선입선출
GUI (Graphical User Interface) 도형사용자대면부
I/O (Input/Output) 입출력
IBM (International Business Machines Corporation) 국제사무기계회사
IPC (Inter Process Communication) 프로쎄스간통신
JCL (Job Control Language) 일감조종언어
LAN (Local Area Network) 국부망
LIFO (Last In First Out) 후입선출
LRU (Least Recently Used) 최대미사용
MVS (Multiple Virtual Storage) 다중가상기억기
NTFS (NT File System) NT파일체계
NUMA (Nonuniform Memory Access) 불균일기억기접근
ORB (Object Request Broker) 객체요청중개기
OSI (Open Systems Interconnection) 개방형체계호상련결
PC (Program Counter) 프로그람계수기
PSW (Processor Status Word) 처리기상태단어
PCB (Process Control Block) 프로쎄스조종블로크
RAID (Redundant Array of Independent Disks) 독립디스크여분배렬
RISC (Reduced Instruction Set Computer) 축소명령모임콤퓨터
RPC (Remote Procedure Call) 원격프로쎄스호출
SMP (Symmetric Multiprocessing or Symmetric Multiprocessor) 대칭형다중처리 또는 대칭형다중처리기
SPOOL (Simultaneous Peripheral Operation On Line) 직결동시주변조작
SVR4 (System V Release 4) 체계 V 공개 4
TLB (Translation Lookaside Buffer) 변환미리보기완충기
W2K Windows 2000